Rapport de soutenance finale

Vroom Studios
présente
Plein Gaz

AURIAU Florian BAUDRON Paul
BESSON Paul CADILLON Jonas

Sup – C1

22 juin 2005

Table des matières

Introduction

Dans le cadre du projet informatique, il nous a été demandé de faire une application sous Delphi et éventuellement un jeu. Nos affinités concernant le sport automobile nous ont rapidement accordé et orienté vers un jeu de voiture de type "Mario Kart". Dans un premier temps nous pensions faire un simulateur de rallye, mais après quelques recherches concernant la programmation et surtout un entretient avec des étudiants de Spé nous nous sommes réorientés vers un jeu arcade plus simple en terme de réalisme automobile et de graphisme.

Ainsi, nous aimerions créer un jeu de course où il serait possible d'intéragir avec les adversaires grâce à des items présents sur le circuit (projectiles, booster ...). Il sera possible pour l'utilisateur de créer ses propres circuits. Le jeu comportera un mode multijoueur permettant à deux joueurs de s'affronter simultanément sur le même écran, ainsi qu'un mode réseau offrant la possibilité de s'affronter sur plusieurs machines en même temps. Le mode solo, proposera d'affronter d'autres adversaires contrôlés par l'ordinateur.

Ce projet sera aussi l'occasion de travailler en groupe, notamment en se fixant des échéances et en respectant ce cahier des charges.

1 Rappel du cahier des charges

1.1 Présentation du groupe

Florian Auriau : Florian vient de l'orne, il pratique l'informatique depuis qu'il a 12 ans. Il a quelques connaissances en HTML et PHP, mais n'a jamais fait que du langage orienté WEB. Grâce à ce projet, il pourra découvrir une autre vision de l'informatique dont il n'a été qu'un simple utilisateur.

Paul Baudron : Originaire du loiret, Paul s'intéresse à l'informatique depuis qu'il a 13 ans. Il n'a pour ainsi dire jamais codé, mais s'est déjà penché sur le sujet à travers des tutoriaux trouvés sur le net. Ce projet sera l'occasion pour lui aussi de découvrir de nouvelles facettes et d'acquérir de nouvelles connaissances dans le domaine des jeux video et plus surtout de la programmation.

Paul Besson : Originaire du Pays d'Aix en Provence, Paul prend rapidement goût à l'informatique. Il fait ses débuts sur de vielles machines et c'est en 1998 qu'il obtient son premier ordinateur et découvre de surcroît l'Internet et ses joies. Pendant ces dernières années il s'adonne à la création musicale, graphique et de site Web. Il découvre la programmation en réalisant un outil permettant de récupérer des données d'un périphérique (sonde thermique et station météo) sur port série. L'année passée à Epita lui a permis de comprendre les mécanismes de la programmation ce qui lui permettra d'aborder sereinement ce projet.

Jonas Cadillon : Jonas vient de Dax dans les Landes, il a débuté l'informatique à l'âge de 12 ans. Il a quelques bases de programmation en Delphi, ainsi qu'en HTML. Grâce à ce projet, il pourra mieux comprendre le fonctionnement d'un moteur physique et de l'intelligence artificielle dans un jeu vidéo.

1.2 Présentation du projet

1.2.1 Choix

Le choix du projet étant relativement libre, comme la plupart des étudiants, nous avons choisi de développer un jeu vidéo. Toutefois, nous voulons nous baser sur un type de jeu existant tout en ajoutant une touche personnelle.

1.2.2 Nature et origine

Après avoir passé en revue bon nombre de genre de jeux vidéo, nous nous sommes fixés sur la simulation de course de type arcade, c'est à dire la possibilité d'utiliser des

items tout au long des courses. Nous n'esperons pas égaler les célèbres "Mario kart" et "Crash Team Racing", mais nous voulons essayer de créer un jeu interactif et divertissant. Les graphismes, comme les voitures seront simplistes et caricaturaux, proche du monde enfantin. Ceci nous permettra de nous concentrer particulièrement sur le principe même du jeu, son interactivité, tout en créant un univers joyeux.

1.2.3 Intérêt

Ce projet présente plusieurs avantages. Tout d'abord, nous apprendrons à chercher des informations par nous même sur la programmation en Delphi et sur l'ensemble des outils que nous allons utiliser. De plus, nous acquierrons des connaissances dans le domaine de la 3D, de la création d'un moteur physique, de la gestion de l'intelligence artificielle des ennemis, etc ... C'est également l'occasion d'apprendre à coordonner un projet dans un temps imparti, ce qui n'est pas toujours une chose simple.

1.3 Découpage du projet

1.3.1 Aspect graphique

Pour le graphisme, en fait ce qui concerne l'esthétique du jeu, nous allons lui créer son propre style, se rapprochant d'un certain point de vue du monde enfantin avec des formes et des couleurs joyeuses, agréables à voir. Pour garder un même esprit stylistique tout au fil des mois de réalisation du projet, nous allons faire un site se rapprochant du thème graphique du jeu en attendant sa finalisation graphique définitive du mois de juin

1.3.2 Aspect sonore

Peut-être de moindre importance en terme de travail, elle devra permettre aux joueurs de se sentir un peu plus immergés dans la course. Il est aussi prévu de récupérer dans des banques de sons libres des effets sonores spéciaux tels que des freinages, les klaxons, et tous les bruits provenant de la mécanique, notamment lors des accélérations accentuant l'immersion du joueur dans la partie.

1.3.3 Moteur graphique

Le moteur graphique affichera l'environnement du jeu, les véhicules, les items, de manière générale l'ensemble des objets qui doivent apparaître au joueur durant la partie. La camera sera placée par défaut au-dessus de la voiture contrôlée par le joueur en question, style vue à la troisième personne, mais l'utilisateur pourra choisir d'autre vue durant la partie.

1.3.4 Gestion physique

Le moteur physique devra gérer tous ce qui concerne les objets en mouvement sur la carte, les chocs entre les joueurs ainsi qu'avec le décor, la trajectoire des projectiles. Il s'occupera de la physique propre à chaque véhicule, c'est à dire de l'accélération, du freinage, des dérapages.

1.3.5 Intelligence artificielle

L'intelligence artificielle devra servir à contrôler les adversaires. Ils devront avoir un comportement à peu près cohérent et être capable de se servir des divers items présents sur la piste. Les concurrents ne seront pas imbattables, c'est pourquoi ils pourront parfois commettre des erreurs.

1.3.6 Mode multijoueur

En mode multijoueur, il sera possible d'affronter un autre joueur sur le même écran mais coupé en deux. L'intelligence artificielle devra tout de même encore contrôler quelques concurrents, pour que la course conserve son intérêt.

1.3.7 Mode réseau

Le mode réseau permettra à plusieurs joueurs de s'affronter en même temps. Les utilisateurs pourront définir le nombre de concurrents gérés par l'intelligence artificielle.

1.3.8 Interface utilisateur

L'interface utilisateur permettra au joueur de naviguer dans les différents menus du jeu. Il aura notamment la possibilité de choisir entre le mode Solo ou Multijoueur, de régler le volume sonore du jeu, ou encore de créer ses propres circuits grâce à un éditeur de map.

1.3.9 Site web

Le design du site web ressemblera fortement à celui du jeu vidéo. Ce site sera réalisé en PHP, nous mettrons à la disposition des internautes, le cahier des charges, les différents rapports de soutenance ainsi que les dernières versions du projet. Il y aura également une gestion des news, qui tiendra au courant les visiteurs de l'avancement du projet, des problèmes rencontrés par le groupe, etc ...

1.4 Distribution des tâches et planning

1.4.1 Distribution des tâches

	AURIAU Florian	BAUDRON Paul	BESSON Paul	CADILLON Jonas
Moteur 3D	X		X	
Moteur physique	X	X		X
Intégration du son			X	
Editeur de map		X	X	
Gestion des entrées sorties		X		
Intelligence artificielle	X	X		X
Interface utilisateur (GUI)		X	X	
Multijoueur	X			X
Modélisation 3D / Graphismes			X	X
Mode réseau	X	X	X	X
Site Web	X			X
Assemblage général du projet	X	X	X	X

1.5 Planning de réalisation

1.5.1 1re soutenance :

- Recherches et documentations générales sur Delphi-OpenGL
- Ebauche et réflection sur le moteur 3D
- Ebauche et réflection sur le moteur Physique
- Ebauche de l'éditeur de map
- Site web commencé

1.5.2 2e soutenance :

- Moteur Physique en cours (collisions, accélération,items ...)
- Moteur 3D en cours (actions standard, vue camera ...)
- Réflection sur L'IA
- Ebauche de la modélisation 3D
- Editeur de map en cours de finalisation
- Ebauche du GUI
- Présentation du site web
- Gestion des contrôles (entrées / sorties) en cours

1.5.3 3e soutenance :

- Ebauche du mode multijoueur
- Début d'implémentation de l'IA
- Gestion des contrôles (entrées / sorties) achevée
- Gestion du son
- Modélisation 3D en cours de finalisation
- GUI finalisé
- Editeur de map finalisé
- Ebauche et commencement du mode réseau

1.5.4 Soutenance finale :

- Finalisation des graphismes
- Finalisation du moteur physique et de la gestion des items
- Finalisation du mode réseau
- Fin d'implémentation de l'IA
- Assemblage général du projet / Débugage
- Manuels / installeur / desinstalleur

1.6 Moyens mis en œuvre

1.6.1 Outils matériels

	Auriau Florian	Baudron Paul	Besson Paul	Cadillon Jonas
Carte mère	Asus A7N8X-X	Asus P4P800 Deluxe	Msi KT4 Ultra	Asus P4C800-E Deluxe
Processeurs	AMD Athlon Xp 2400+	Pentium 4c 3 Ghz	AMD Xp 2800+	Pentium 4 3.5 Ghz
RAM	512 Mo DDR PC 3200	1024 Mo DDR PC 3200	1024 Mo DDR PC 3200	1024 Mo DDR PC 4400
Carte Graphique	Ati Radeon 9600 Pro	Ati Radeon 9800 Pro	Nvidia Geforce 5900XT	Nvidia Geforce Ti 4200
Disque Dur	160 Go	160 Go	160 Go	440 Go
Moniteur	Philips 17	LG Flatron 19	Mitsubishi 19	Iiyama 19
Portable	Oui	Non	Oui	Oui

1.6.2 Solution logicielle

En plus de l'aspect matériel que nous avons evoqué précédemment nous aurons besoin d'outils logiciels. En effet nous avons choisi de coder notre projet en Delphi 7 avec l'aide des librairies OpenGL en ce qui concerne l'affichage 3D, et Fmod pour ce qui est du son. De même nous intégrerons au jeu des modèles 3D, probablement modelisés sous Cinema 4D XL R8 et 3DStudioMax. Les graphismes 2D seront eux conçus à l'aide de Adobe photoshop CS. Le site Web sera realisé en PHP à l'aide de Edit Plus en ce qui concerne le code HTML. Bien évidemment Delphi reste l'outil majeur, qui nous permettra de coder le jeu en lui même.

1.7 Aspect économique

Description	Quantité	Prix
Licences Windows XP Pro (OEM)	4	760 €
3D Studio Max 6	1	4900 €
Delphi 7 Entreprise	4	2390 €
Cinema 4D XL R8	1	2690 €
Adobe Photoshop CS	1	1790 €
PC Portable	3	4399 €
Switch 5 ports 10/100	1	27,99 €
	Total :	16956,99 €

En regardant le tableau ci dessus on constate que ce projet est très honéreux pour des débutants. Cela représente donc une motivation supplémentaire à sa réussite ! Cela dit, il est normal que des merveilles telles que Photoshop soient payantes, afin de remercier ses créateurs, même si c'est l'entreprise qui en profite ... Néanmoins, nous aurons au moins la "fierté" de n'utiliser que des logiciels dont nous possèdons la license ...

2 La construction du projet point par point

2.1 Moteur 3D [baudro_p] [besson_p]

2.1.1 Introduction

L'affichage graphique est la partie la plus importante du projet car c'est la plus visible du jeu. Il faut donc implementer des algorithmes rapides pouvant traiter un maximum de données. De plus il faut donner un style au jeu, un univers spécifique et des couleurs rappelant le monde enfantin conformément au cahier des charges initial.

2.1.2 Avec le style ...

Comme je l'ai dit un peu plus haut, le jeu doit avoir un style enfantin. Nous l'avons conçus avec des couleurs primaires et des formes simplifiées. La recherche stylistique a débutée dès la première soutenance en créant notament les premiers modèles en 3D et le site web. On constate que la couleur jaune, agréable à l'œil ,est présente dans tout le jeu. Les véhicules sont inspirés des jouets en bois pour enfants.

2.1.3 Fenêtre Opengl

Après s'être posé la question du choix de la librairie graphique **Opengl** du jeu, il fallait initialiser la fenêtre opengl. Pour cela, nous nous sommes tournés vers glfw car

il est tout de même plus simple que opengl pour l'initialisation d'une fenêtre. Il y a aussi glut, cependant plus lent que glfw, notre choix a donc été glfw. Pour coder avec OpenGL sous Delphi, nous avons donc choisi les librairies :
- OpenGL
- Glu, surcouche avancée d'OpenGL qui permet la construction d'objets complexes, la gestion de caméras par exemple, le dimensionnement d'images, l'alpha blending mais aussi la création de mini cartes (récupération de sa position sur le circuit).
- Glfw pour l'initialisation de la fenêtre OpenGL.
- Glut, qui, comme Glu, complète la palette des fonctions OpenGL.

Avant d'afficher tout objet sur l'écran, il faut charger la librairie OpenGL et créer la fenêtre où les objets évolueront. Nous avons utilisé la librairie Glfw qui se charge de créer la fenêtre OpenGL avec une seule fonction : glfwInit. Il faut aussi spécifier notamment les dimensions de la fenêtre avec glfwOpenWindow qui créera la fenêtre.

Couleurs Avec OpenGL on applique les couleurs à chaque point du polygone. On détermine une couleur avant de mettre les coordonnées du point, et l'interface 3d de l'ordinateur s'occupe du traçage.

Primitives Comme vous le savez sûrement si vous avez légèrement touché à la 3D, tous les objets sont en fait une association d'éléments de base : ce sont des faces orgranisées dans un certain ordre dans l'espace, qui représentent l'objet. Chaque face peut être décomposée en une série de facettes triangulaires. Ainsi, un rectangle est en fait constitué de deux triangles consécutifs. Ces triangles sont eux-mêmes composés de 3 cotés, ou 3 lignes, et de 3 points (ou vertex) qui définissent les extrémités de ces segments. Ces éléments (points, lignes, triangles) sont aussi appelés primitives. OpenGL est basé sur le principe des primitives, c'est-à-dire que pour dessiner un objet, il faut dessiner toutes ses primitives. Et pour cela, on utilise des fonctions très simples d'utilisation et permettant des dessins très poussés, puisque grâce à ces fonctions, on peut dessiner quasiment tout.

Passons maintenant à la pratique : comment dessiner un élément primitif ? On appelle glBegin() avec comme paramètre l'élément, on lui envoie les coordonnées des vecteurs-clés via glVertex(), et on termine avec glEnd(). Notez que vous ne verrez rien si vous n'appelez pas SwapBuffers(DC). On utilise une technique appelée double-buffering, qui consiste à afficher une image pendant que l'on calcule l'autre, et lorsque celle-ci est terminée, on échange les deux. Donc si on enlève SwapBuffers(DC), on verra toujours un écran vide, tandis qu'on dessinera dans un écran non visible. Cf cubeblanc et code.

C'est vrai que nous avons un peu de mal à faire la distinction entre l'une ou l'autre face. C'est pour cela que j'ai choisi de mettre des couleurs aux faces. Nous en venons à la coloration grâce à cette merveilleuse fonction glColor() ! Comme pour glVertex(), glColor() va de glColor3b() (R,G,B sur un octet signé) à gl- Color4usv() (R,G,B,Alpha sur un pointeur sur short non signé) en passant par glColor3d() (R,G,B sur un double, de 0.0 à 1.0). C'est justement cette dernière fonction que nous allons utiliser car pouvoir

sélectionner l'intensité de la couleur entre 0 et 1 est bien pratique. Pour les codages sur des valeurs entières, tout l'intervalle disponible est utilisé. En fait, quand on appelle glColor, la couleur des sommets que l'on dessinera après l'appel de la fonction changera, selon le principe de la machine d'état. Donc puisqu'on veut une couleur unie par face, on va appeler glColor() 6 fois, une fois avant chaque face. Cf Cubecouleur et code.

Les couleurs sont basiques, mais là au moins on voit la différence entre les faces.

2.1.4 Affichage de la map

Le moteur d'affichage de map est doté d'un *quad-tri* permettant de gagner environ 1/3 du temps pour afficher une image (35 pourcent de fps supplémentaires) par rapport au moteur initiale sans optimisation. Ici on dessine l'environnement autour de la map en permanance et au centre, on recupère la position du véhicule (grâce au type *voiture*) à chaque image et on affiche uniquement les morceaux de circuits concernant la voiture. On peut changer le coefficient pour avoir une profondeur de champ suppérieur, mais en l'état actuel des choses on gagne déjà suffisament d'images par seconde pour adopter ce mode d'affichage de la map. On peut augmenter le coefficient de proximiter jusqu'a obtenir une augmentation de la moitié du taux d'image par seconde.

2.1.5 Fichiers 3DS et Chargement . . .

Le convertisseur a pour but d'importer des objets 3DSMax et de les transformer en fichier exploitable ensuite par le moteur graphique. Cela à 2 avantages conséquents :
- La création simple d'objets complexes (primitives)
- L'obtention des données selon notre souhait (position)

Voir annexe pour le code.

Les fichiers d'origines sont de type ASCII (format d'export de 3D studio Max). Premièrement, le programme va chercher dans ce fichier la liste des faces et l'enregistre dans un tableau. Ensuite, le programme accède aux coordonnées des vertex de chaque faces qui sont enregistrées dans le fichier résultat. On obtient une liste de coordonnées des vertex. La texture appliquée à l'objet est stockée dans la première ligne du fichier.

Recherche Nous avons repris les sources d'un loader de fichiers trouvé sur Internet, mais les principaux défauts de ce loader sont que d'une part il est extrêmement complexe, il prend en compte tous les effets (lumière, ombre, etc), d'autre part, il provoquait énormément de conflits entre les procédures utilisées pour les textures du projet et celle utilisée par le loader 3ds, il y avait aussi des conflits au niveau d'enregistrements qui étaient déclarés plusieurs fois. Afin de faire correctement le loader, nous avons du effectuer des pas a pas de toutes les fonctions qu'il utilise afin de comprendre comment il fonctionne, le problème c'est qu'il utilise des classes définies et nous n'avons pas encore compris leurs fonctionnements. Nous avons du modifier le loader, car il est était conçu

à la base pour loader qu'un seul modèle ainsi nous avons créé un tableau de modèles, T3DModel.

Ensuite pour placer les modèles aux endroits voulus, il n'y a juste qu'à utiliser la fonction Gltranslate et éventuellement des Glrotate.

Pratique Nous avons implémenté une procédure permettant de charger un model de type 3DSMax à la position souhaitée en x, y et z. Par la suite il est aisé de modifier son orientation avec la fonction glRotate.

Cette procedure :

On peut afficher tous les models destinés au jeu. Aussi bien la map que les bolides et c'est à l'aide du moteur physique qui à chaque tour de boucle incrémente les positions en x ,y que les objets s'animent dans notre fenêtre openGL. En fait chaque model est en fait stocké dans un tableau de T3DModel.

2.1.6 Optimisations

Suite à l'ajout important de modèles 3D, de textures, le taux d'images par seconde a beaucoup baissé, il a donc fallu trouver un moyen pour pallier ce manque plus que gênant, d'où l'utilisation de listes d'affichages pour la skybox qui ne change pas pendant le jeu et d'un affichage de la map avec un Quad-tri. Des fonction inherentes à opengl permettent de ne pas afficher les faces non visibles des polygones, elles sont utilisée, leurs gains en performance est cependant faible.

2.1.7 Conclusion

Depuis la premère soutenance notre vision de l'affichage pour notre jeu est restée la même et est maintenant parfaitement aboutie notamment grâce aux divers révisions de nos algorithmes et à nos optimisations.

2.2 Editeur de Carte [baudro_p]

2.2.1 Son Rôle

L'éditeur de carte permet de créer des circuits en 2 dimensions, il se présente sous la forme d'un quadrillage de 9 cases par 9. Il dispose des fonctionnalités d'enregistrement et d'ouverture de fichiers qui doivent être en format texte. Pour plus de facilité, on pose l'extension des fichiers en .map, ce qui permettra par la suite de les repérer plus rapidement pour leurs chargement dans le jeu par exemple.

Cet éditeur nous permet donc de faciliter la création de carte, ce qui aurait été très fastidieux si il n'aurait pas été là. Il sera, bien évidemment, utilisable par les utilisateurs finaux de notre jeu (si il y en a...).

2.2.2 Evolutions et Caractéristiques

Au cours de l'avancement du projet, l'éditeur c'est vu affubler de nouvelles options pour en faciliter son utilisation, comme par exemple le chargement et l'enregistrement via des fenêtres windows, des refontes graphique et ergonomique,. . .

Donc graphiquement la carte est représentée par un quadrillage de 81 cases, mais en mémoire c'est un vecteur de chaine de caractère de 81 cases.

Mais son principe général n'a pas changer, l'éditeur fonctionne de la manière suivante :

(1) Il initialise chaque image du quadrillage sur une valeur de case vide.

(2) Lorsque l'on clique sur l'un des objets à droite, une variable temporaire reçoit la valeur du type de la case cliqué.

(3) Ensuite lorsque l'on clique sur une des cases du quadrillage la valeur temporaire est assignée à l'élément du tableau qui correspond à la case cliquée.

(4) Pour l'enregistrement, c'est une boucle qui passe en revue toutes les valeurs du tableau (du 1er élément au 81e élément) et qui les écrient au fur et à mesure dans un fichier.

(5) Pour l'ouverture, c'est aussi une boucle qui lit toutes les lignes du fichier map et qui les stocke dans les éléments du tableau, ensuite chaque image du quadrillage reçoit la valeur du tableau lui correspondant.

cf. EditMap1

L'aspect visuel a beaucoup évolué mais le principe de choisir son morceau de circuit à droite et de cliquer sur le cadrillage de gauche pour l'appliquer a toujours été le même.

De nouvelles options sont apparues ensuite. Tout d'abord, le choix entre 2 modèle différents pour le même type de case ce fais grâce à un ComboBox(1). Ensuite le chargement et l'ouverture des cartes se font via des fenêtre windows(2), ce qui facilite l'utilisation pour les futures joueurs.

La ComboBox va seulement intervenir lors du "cliquage" sur les images de selection, si l'on clique et que le morceau selectionnée est du type 1, se sera une chaine de caractère différente qui sera enregistrée dans le tableau et finalement dans le fichier.

cf. EditMap2 - EditMap3

De plus, l'éditeur possède une fonction qui empèche les utilisateurs d'enregistrer des circuits qui ne "tourne pas rond", cette fonction est appellée quand on tente d'enregistrer le travail, si le circuit est mauvais un message nous informe du problème et l'on retourne sous l'editeur.

cf. EditMap4

Cette vérification s'effectue de la manière suivante :
–en premier lieu on va chercher le départ et vérifier qu'il est unique avec une boucle qui passe en revue toutes les cases du vecteur qui représente la carte.
–ensuite si le départ et présent on va lancer une autre boucle qui va suivre le circuit et qui va vérifier que chaque passage d'un bout de circuit à un autre et valide.

Dans cette boucle on va proceder comme cela :
-on retient la position de la case précedente
-avec les position de la case actuelle on connait quelle type de case elle représente et donc quelle "sortie" elle doit avoir (2 sorties par case)
-avec la position précédente on sait par où on est arrivé, on a donc plus qu'une sortie qui nous donne la position de la case suivante à tester
– si cette case suivante se "raccorde" avec la case courante alors on passe à la suivante (en avançant aussi la case précédente)
– si elle ne peut pas se "raccorder" alors un booléen de continuiter passe à faux et on sort de la boucle "tant que"
-la boucle "tant que" s'exécute tant que l'on est pas retombé sur le départ (on le test avec les positions de la case) ou que le booléen de continuiter et à vrai
-finalement on regarde à cause de quoi on est sortie de la boucle, et on a notre réponse.

Finalement, l'éditeur avez besoin d'un paramétrage "manuelle" pour connaitre sa position sur le disque, ce positionnement ce fait par une demande simple à l'utilisateur via une fenêtre d'ouverture windows si l'éditeur n'arrive pas à se situer lui-même.

Le chargement des cartes se fait de manière automatique, le jeu charge les cartes qui sont présentes dans le dossier "map" qui est lui même dans le répertoire du jeu (les cartes doivent avoir une extension en .map). Si aucune carte n'est présente dans ce dossier alors le jeu refuse d'aller plus loin que le menu général et affiche un message d'erreur, et la seule chose que l'on peut faire et de cliquer sur quitter.

Cet éditeur et donc plutôt simple à utiliser, même si il ne fait que des circuit en 2 dimensions, il est complet dans ses fonctionalitées, les joueurs pourront ainsi créés de nombreux circuits et de manière assez rapide.

2.3 GUI [baudro_p]

2.3.1 Son Rôle

Le GUI (General User Interface) permet de faciliter le lancement ainsi que la gestion des options pour l'utilisateur. Il se presente sous forme de bouton à cliquer pour passer d'une fiche à une autre.

C'est lui qui est lancé en premier, il permet de naviguer entre les différents modes de jeu, de configurer les contrôles, la résolution,... et finalement de lancer la partie.

Il centralise toutes les informations du projets et les redistribue pour initialiser une partie, par exemple, l'utilisateur va lui dire combien de joueur contrôlé par l'Intelligence Artificielle il veut avoir dans son jeu, sur quelle carte il souhaite jouer, par quelle modèle il veut être représenté,... et le GUI va lancer une partie avec tous les paramètres adéquates. De la même manière, il va permètre de rentrer l'adresse IP du server pour s'y connecter lors de partie reseau.

2.3.2 Principe

Le principe générale du GUI est resté le même tout au long de l'avancement du projet, juste quelques détails et son aspect graphique ont changé.

Tout d'abord, il faut savoir que le GUI est créé par plusieurs fiches, ou pages. Ces fiches sont dessinées que si l'utilisateur l'a demandé. Ses demandes sont faite via des boutons qu'il peu cliquer pour naviguer dans le GUI.

Le GUI est basé sur une boucle "tant que" qui s'exécute tant que l'on a pas cliqué sur le bouton "Quitter". Pour chaques pages du GUI, un booléen lui est assigné, c'est lui qui va nous permettre de savoir si une page doit être affichée ou non et si les boutons liés à cette page doivent être eux aussi testés ou non.

Pour afficher la bonne page, des conditions sur les booléens des pages vont être testées à chaque tour de boucle :
-si un booléen est à vrai alors la page lui correspondant est affichée et les tests des boutons pour cette page sont effectués.

-lorsque l'on clique sur un bouton, les booléens des pages sont modifiés pour afficher la bonne page au prochain tour de boucle et pour faire les nouveau test sur le boutons.

Les boutons sont gérés comme des zones dans la fenêtre OpenGL. A chaque tour de boucle, on recupère la position de la souris en x et y grâce au composant GLFW et lorsque elle se situe dans la zone d'un bouton on modifie le celui-ci pour montrer à l'utilisateur qu'il se situe sur un bouton (comme un rollover) puis si il clique tout en étant dans la zone, on modifie les booléens des fiches pour afficher juste celle qu'il faut.

Par exemple, pour charger la page de la configuration des touches, il faut que la position en x de la souris soit entre x_1 et x_2, de même pour la position en y, et si le bouton gauche de la souris est appuyer alors que ses position vérifie les conditions précédentes, les booléens sont modifier,...

cf. GUI1

Cette méthode marche trés bien si la résolution ne bouge pas, car les positions des boutons sont enregistrées en "dur" dans le code. Mais à partir du moment ou l'on change la résolution les positions de la souris sont différentes, elles peuvent être beaucoup plus grandes ou plus petites. Il a fallu donc changer le type de récupération de la position de la souris. La position est maintenant calculée en fonction de la résolution et les boutons sont eux repérés suivant des position en pourcentage, ce qui a pour effet de comparer la position, en pourcentage, de la souris avec des positions, en pourcentage, pour les boutons, ces tests sont donc toujours bon quelque soit la résolution.

Ensuite, Le texte affiché n'est plus "statique" (la première version affichée des images) mais "dynamique", une procédure d'affichage de texte a été implémenté pour alléger le code et pour faciliter la gestion du GUI, elle lit caractère par caractère et crée un `GL_QUADS` avec la bonne partie de la texture qui récapitule les lettres ou les chiffres, la texture est déclinée en deux version, une pour le texte bleu et une autre pour le texte rouge (les chiffres ont une couleur unique). Le `GL_QUADS` ainsi que les espacements sont modifiés en fonction de la taille d'affichage demandée en paramètre. Ce texte ainsi affiché peut être, et est utilisé en pleine partie pour faire passer des informations aux joueurs.

La version finale du GUI permet ainsi d'ajouter des fiches et des nouveaux boutons de manière assez rapide et simple, elle permet donc d'intégrer rapidement les ajouts ou les modifications liées au projet tant que celles-ci ne sont pas trop importantes dans l'architecture générale du projet.

cf. GUI2

2.4 Moteur Physique [baudro_p]

2.4.1 Son rôle / Spécifications

Le moteur physique va permètre de fixer des rêgles dans le monde que nous avons tenté de créer, c'est lui qui va s'occuper des collisions avec les décors, les voitures présentes sur le circuit, les missiles ainsi qu'avec les objets deposés sur le sol. Il centralise les information les traitent et les passe au moteur suivant qui s'occupera de sa tache.

Le moteur physique reçoit en entrée toutes les informations concernant l'enssemble des entitées, c'est-à-dire, leurs positions, leurs orientations, leurs accélérations,... mais aussi les actions que les joueurs veulent exécuter pour cet appel du moteur.

Le moteur va donc répondre aux attentes des joueurs en bougeant leur véhicule dans la direction désirée, dans la mesure du possible car il doit aussi tenir compte des décors qui ne peuvent pas ètre traversé. En effet lors de la rencontre avec un décor, une voiture va subir une force de répulsion, elle va rebondire, et de la même manière avec les autres voitures.

De plus le moteur physique gère quelques effets comme l'accélération, la décélération... Il gère aussi l'enssemble des objets, leur déplacement si besoin (pour les missiles par exemple),les collisions avec eux et les joueurs.

2.4.2 Evolution

Pour obtenir notre moteur physique finale nous somme passés par de nombreuses versions qui se sont améliorées, et se sont vu dotée de nouvelles fonctionnalitées, au fur et à mesure de l'avancement du projet.

Les premières versions été basée sur le déplacement verticale et horizontale, mais cette méthode ne correspondait pas à notre type de jeu qui se voulait ètre avec une vision "suivie" ou à la troisième personne, et c'est via l'inclusion d'un angle que tous les déplacement se calculent maintenant.

2.4.3 Les différentes versions

-1re Version Les objets sont caractérisés par leurs positions en x et y(3), ainsi que par leur accélération suivant ces deux axes.

Le moteur physique ajoute à chaque appel, l'accélération à la position sur l'axe lui correspondant.

L'accélération suivant un axe est augmentée ou diminué si l'on appuie sur la touche pour aller dans vers le sens positif ou négatif de l'axe (x ou y).

Les rebonds sont simulés en remplacant l'accélération (en x ou y) de l'objet par son opposée(1), lorsque l'objet dépasse une certaine limite repéré avec sa position en x ou y.

Les collisions entre les objets sont détéctées grâce à la distance(2) les séparrant, si cette distance est trop petite alors on interchange les accélérations en x et en y des deux protagonistes, ce qui simule une force de répulsion.

cf. MoteurPhysique1

-2e Version Dans cette version nous avons introduit un angle pour chaque entitée qui se déplace, qui permettra une future gestion des rotations de la caméra, des modèles,...

cf. MoteurPhysique2

Pour ce qui est des collisions entre les voiture, le principe est toujours le même, une distance qui est calculée pour savoir si les deux voitures sont en collision. Ces testes sont effectués pour toutes les voitures qui sont stockées dans un vecteur de voiture. Si il y a en effet collision, les angles d'orientation sont interchangés et les accélérations aussi.

Nous avons aussi intégré les collisions avec les objets immobiles, si une voiture collisione avec une bombe posée sur la piste par exemple, un timer est lancé pour que la voiture concernée ne puisse plus rien faire pendant un certain temps, ce qui correspond au malus engendré par la collision. Une fois que la collision a eu lieu et que les conséquences on été appliquées l'objet est effacé du tableau.

Et finalement, les collisions avec les missiles, ce test est aussi réalisé via une boucle qui parcours la liste des voitures et des missiles et qui modifie lorsqu'il y a collision, la vitesse de la voiture concernée et qui la bloc pendant un certain temps (de la même manière que pour les objets immobiles), le missile est effacé de la liste pour ne plus interférer avec les autres voitures aux tours de boucles suivants.

Ensuite, la modification des caractéristiques de la voiture du/des joueur(s) est gérer suivant les touches appuyées ou non. On regarde si les touches sont appuyées ou non que si la voiture n'est pas bloquée à cause d'un missile ou d'un objet immobile.

-pour tourner : si une des touches est appuyées, on incrémente ou décrémente l'angle d'accélération avec un modulo de 2π.

-pour avancer : l'accélération générale de la voiture est augmentée et ses positions selon x et y sont recalculées en leur ajoutant l'accélération multipliée par cos(angleDeLaVoiture) pour les x et l'accélération multipliée par sin(angleDeLaVoiture).

-pour reculer : l'accélération générale de la voiture est décrémentée et les positions sont recalculées de la même manière que précédemment.

-si aucune touche n'est appuyer : alors la voiture doit décélérer, ce qui est traduit pas le fait de diviser l'accélération générale par une constante jusqu'à ce qu'elle soit trop petite et finalement réinitialisé a zéro. Les position sont recalculées comme précédemment.

-pour utiliser l'objet : si la touche correspondante est appuyée et que le joueur a bien un objet à utiliser alors une procedure de création et de lancement est exécutée.

Pour le déplacement des missiles, qui est une action indépendante du reste, une boucle se charge de passer en revue le vecteur de missile et de vérifier si il faut les déplacer. On utilise leur accélération qui est constante pour modifier leur caractéristiques, comme pour les voitures.

Finalement, une des grosse étape, les collisions avec les decors. Il faut savoir que lors de l'enregistrement avec l'éditeur de carte chaque case c'est vu attribuer un type que l'on pourrai appeller "masque".

cf. MoteurPhysique3

Quand la carte est chargé en mémoire il est facile de savoir sur quel type de case un joueur se situe (avec un div 10 des positions en x et y car une case fais 10 par 10 en taille) et ensuite on peut savoir où sur la case il se situe (avec un mod 10 des positions en x et y) On peut ainsi faire les tests suivant les cases (suivant les "masques") et donc modifier les caractéristiques de la voiture en fonction de sa position sur une case.

Pour résumer on se sert de chiffres des dizaine pour savoir sur quelle case la voiture se situe, et des chiffres de unitées pour savoir a quelle endroit sur la case la vioture se trouve.

Pour conclure, le moteur physique est assez générale pour ètre appliquer a tous les types d'objet, par exemple l'Intelligence Artificielle utilise le moteur physique de manière autonome et sans aide autre que ses fonctions ou procédures propres.

2.5 Entrées-Sorties [baudro_p]

2.5.1 Son rôle / Spécifications

La gestion des entrées-sorties est nécessaire à l'utilisateur pour qu'il puisse transmettre les actions qu'il désire efféctuer à l'ordinateur. Cette gestion ce fait juste avant le moteur physique, ce qui lui permet de savoir exactement ce que le joueur veut faire.

Les entrées-sorties sont limitées au clavier, toutes les touches "standards" sont utilisables et configurables via une page dans le GUI.

Cet gestion peut ce résumer à : connaitre l'état d'une touche à un instant précis et efféctuer la ou les action(s) qui lui sont attribuées, par exemple si l'on désire avancer, il suffit d'appuyer sur la touche correspondante, qui a été configurée dans le GUI, et l'enssemble des procédures et fonctions qui sont nécessaires pour faire avancer une voiture seront executées.

2.5.2 Principe

Depuis le début du projet, c'est le composant GLFW qui permet grâce à une de ses fonctions de connaitre l'état d'une touche, c'est-à-dire si elle est appuyée ou pas. Cette fonction nous renvoit donc un booléen, et via une série de test on peut connaitre les intentions du joueur pour ce déplacer, tirer un missile,... ces actions seront ensuite reprise par le moteur physique pour donner suite aux déplacements, et aux collisions si nécessaire.

Pour ce qui est de la configuration des touches dans les GUI, deux méthodes ont été implémenté.

La première se voulait semblable à la configuration des touches des jeux actuelles, sont principe été basé sur une boucle "tant que" dans laquelle on faisait varier une valeur entre le nombre min et max équivalent aux deux valeurs ordinales, la plus basse et la plus haute, correspondant aux touches du clavier, cette boucle s'exécutait tant que aucune touche n'était appuyé, de cette manière il aurait suffit de regarder pour qu'elle valeur ont été sortie de la boucle et ainsi connaitre la touche qui été appuyer, donc celle que l'on voulait configurer. Mais cette méthode "bloquait" la fonction GLFW, ce qui ne donnait pas le résultat escompté.

Ce principe à été légèrement modifié pour ne plus faire de boucle infinie (le problème rencontré précédemment), la boucle ne faisait plus qu'un ballayage de toute les valeurs des touches et l'on pouvait obtenir ainsi la valeur de la touche appuyé qui était ensuite enregistrer dans le tableau des contrôle du joueur. Il fallait donc appuyer sur la touche que l'on voulait lier à une action et seulement ensuite cliquer sur le bouton de l'action que l'on voulait lier(pour lancer la recherche de la touche appuyée).

Ensuite, la deuxième méthode est celle qui est dans la version finale. Elle utilise une autre fonction du composant GLFW, elle permet de lancer une fonction, grâce a un pointeur sur foction, dès que l'état d'une touche est modifié, par exemple si l'on relache ou touche ou si l'on appuie dessus. Avec cette fonction on peut recupérer la valeur de la touche appuyée ainsi que son nouvelle état (appuyé ou relaché). Cette méthode se rapproche un peut plus des jeux commerciaux mais ce n'est pas tout a fais la même chose.

Lorsque l'on clique sur une action à lier, une variable retient qu'elle action a été choisi et quand on appuie sur une touche, la fonction appellée par ce changement d'état va enregistrer cette touche pour la bonne action dans le tableau de contrôle du joueur un ou deux.

Les contrôle sont enregistré et chargés de manière automatique suivant les dernières touches configurées lors d'une précédente utilisation du jeu. La configuration est chargée via un fichier qui est lu si il existe et est valide ou qui est créé avec des valeurs par defaut si il n'est pas présent ou invalide. Toutes les touches peuvent donc se voir assignées une action et il est délégué au bon sens de l'utilisateur de ne pas créer des conflits entre les differentes actions et joueur.

Finalement, dans la première version, les touches qui était assigné n'était pas affiché puis les dernières versions se sont vu attribué une refonte graphique, ce qui a permis dans un premier temps d'afficher la valeur ordinale des touches du clavier puis via une fonction de traduction les noms que l'on peu donner au touches les plus utilisées.

cf. EntreesSorties1 - EntreesSorties2 - EntreesSorties3

2.6 Intelligence Artificielle [baudro_p]

2.6.1 Son Rôle

L'Intelligence Artificielle permet de jouer avec des joueurs contrôlés par l'ordinateur. Dans le mode monojoueur et multijoueur en local, on peu compléter le nombre totale

de participant pour en avoir jusqu'à quatre en même temps. Ces bots suivent le circuit de manière frénétique, il utilise les objet un bref instant après les avoir ramassés si ils peuvent toucher quelqu'un avec leur missile par exemple.

2.6.2 Principe

Tout d'abord, l'ajout de l'IA a nécessité la traduction de la carte pour l'ordinateur. Cette traduction c'est déroulée de manière à appliquer un ordre simple à une case donnée. Les ordres sont de quatre types, en fait c'est la direction que la voiture doit suivre quand elle est sur la case.

Le principe de traduction de la carte ressemble à celui de l'éditeur pour ce qui est du parcours et de la condition d'arrêt sur le retour au départ (sauf que là il n'y a pas de booléen pour savoir si le circuit est continue puisqu'il a déjà été enregistré), il suit donc le circuit grâce à la carte créée par l'éditeur et inscrit dans la case correspondante de la matrice l'angle vers lequel la voiture doit tendre pour aller vers le morceau suivant.

cf. IA1

Sachant que chaque morceau de circuit à deux entrées ou sorties possible, il faut savoir comment on est arrivé sur le morceau actuel, on garde donc en mémoire l'angle précédent et ainsi on a plus qu'une seule sortie possible, donc un seul angle possible vers lequelle la voiture doit tendre pour passer à la suite.

Une fois cette traduction achevé, les bots seront exactement quoi faire pour finir le circuit, par exemple :
-quand un bot est sur une case, avec ses position il sait où il se trouve (de la même manière que pour les colisions avec les décors),il va pouvoir connaitre vers quel angle il doit tendre pour sortir de la case et aller vers la suivante.
-une constante sera donc ajouté ou retiré suivant sont angle actuel pour que celui-ci se rapproche de l'angle final recherché.
-ensuite le bot va passer à la case suivante et il répètera cette étape de modification de l'angle.

De plus pour que la voiture soit le maximum au centre de la case, on va lui appliquer un petit recentrage qui va modifier, via une constante ajouté ou retiré, l'angle vers lequel le bot doit aller.

Par exemple, si la voiture est trop sur un coté et que sa direction doit tendre vers $\pi/2$ alors on ajoute, ou l'on soustrait, une constante à $\pi/2$, la constante est différente selon l'endroit où le bot se trouve sur la case. Une fois revenue au centre de la case la constante est remise à 0 et donc la direction redevient celle donnée par defaut pour la case où la voiture se situe.

cf. IA2

Pour ce qui est de l'utilisation des objets ramassés sur le circuit, il faut savoir que "l'heure" du ramassage est enregistrée pour la voiture concerné et que à quelques secondes. Une fois ce temps écoulé le bot peu utiliser sont objet.

Si c'est un accélérateur, ou une caisse explosive, il n'y a rien qui l'empèche de les utiliser à n'importe quel moment, comme un humain en quelque sorte. . .

Mais pour ce qui est des missiles, il faut que le bot ait une cible dans sa ligne de mire. C'est une fonction qui va répondre à cette question. Cette fonction prend en paramètre l'identifiant du bot concerné, avec sa position et son angle d'orientation, on peut calculer un vecteur qui serait le vecteur directeur de la droite suivi par le missile, il suffit ensuite de regarder si ce vecteur est colinéaire avec le vecteur formée par les points ayant comme positions celles du bot et celles de la cible potentiel.

cf. IA3

C'est donc une boucle qui va regarder cette condition pour chaque autre joueur et dès que cette condition est verifiée le bot reçoit l'autorisation de tirer son missile.

2.7 Rêgle du Jeu - Principes [baudro_p]

Le jeu est soumis à quelques rêgles et principes comme le fait de devoir passer sous tous les checkpoints pour valider un tour, ou encore les quelques objet que l'on peut ramasser ou lacher sur la piste,. . .

2.7.1 Les Checkpoints

Au chargement de la carte, une liste des checkpoints est créée pour chaque joueur, elle sera modifiée quand le joueur passera sous un checkpoint qu'il n'a pas encore pris et réinitialisée quand il passera par la case départ si tous les checkpoints ont été pris.

La liste des checkpoints est donc initialisée à une valeur de ”non pris” (valeur négative) puis quand le joueur passe sur un checkpoint une boucle vérifie si il l'a déjà pris (les checkpoints ont un identifiant unique généré avec leurs positions en x et y) et si ce n'est pas le cas le checkpoint est enregistré dans la liste du joueur sur une des valeurs négatives.

Quand le joueur passe sur la case départ, une boucle vérifie si il a pris tous les checkpoints (la liste n'a plus de valeur négative) et valide le tour si c'est le cas.

Son nombre de tour est diminué de 1, et si il passe à -1 alors le joueur est ”bloqué”, ses appels aux fonctions pour le déplacé sont désactivés quand son nombre de tours restant est a -1.

2.7.2 Le classement

Le classement est affiché sur lorsque le joueur a fini tous ses tours de circuits, c'est-à-dire quand son nombre de tour est à -1.

Il affiche les numéros des joueurs et leurs temps lorsq'il ont fini leurs tours.

Le temps afficher correspond à ”l'heure” de fin de course, spécifique à chaque joueur, moins ”l'heure” de départ de la course, qui est enregistrée lorsque l'on démarre juste après le compte à rebour.

Le premier joueur qui a sont nombre de tour restant qui passe à -1, est le premier qui est enregistré dans le vecteur classement, ensuite, dès qu'un joueur passe a -1 une boucle cherche la première place libre dans le classement et l'enregistre, et ainsi de suite, le vectuer se remplis de cette manière.

Lorsque le classement est affiché, le joueur ne peu plus se déplacer, tous les appels aux fonctions de déplacement sont désactivés, comme dit précédemment.

2.7.3 Les caisses à ramasser

Les caisses que les joueurs peuvent ramasser sont situées sur les checkpoints au nombre de 2 par checkpoint. Le test qui va déterminer si un joueur prend une caisse ou non est effectué quand il passe sur un checkpoint en calculant la distance qui le sépare des caisses présente sur celui-ci, si cette valeur est assez petite, il y a collision, la caisse disparaît et si le joueur n'avait pas d'objet à ce moment là, il en a un après.

La caisse dispose d'un champ "temps" qui prend la valeur de "l'heure" où elle est prise et l'empêche d'être affiché tant que le temps courant n'est pas supérieur au temps de la caisse plus trois secondes. Les joueurs ne peuvent bien évidemment pas prendre la caisse tant que celle-ci n'est pas réaffichée.

2.7.4 Les différents objets

Lorsque le joueur rammasse une caisse, il obtient un objet qui peu être de trois type, un missile, une caisse explosive, ou un accélérateur. Cette obtention se fait de manière aléatoire.

Lors de l'utilisation d'un missile, ce dernier est ajouté à sa place dans le vecteur des missiles, un joueur ne peu avoir que un missile, de toute façon les missiles sont assez rapides pour disparaitre en dehors de la carte avant que les joueur aient eu le temps d'en rammaser un autre.

Quand un missile est créé il prend comme direction celle de son lanceur et comme vitesse une constant qui est plus élevée que l'accélération maximale des voiture, pour pouvoir ainsi les rattraper. Le déplacement et les collisions ce font grâce au moteur physique avec une boucle qui déplace tous les missiles si ils existent et calcule les distances séparant les joueurs des missiles, si il y a collision, le joueur concerné est bloqué.

Pour les caisses explosives, elles sont elles aussi ajoutées à leur vecteur, un joueur peut placer jusqu'à 3 caisses, si il en place une 4e, elle remplacera une des caisse qu'il a déjà placé. Les collisions entre les joueurs et les caisses explosives sont traitées via le moteur physique grâce à une boucle qui vérifie la distance séparant une caisse explosive et un joueur. Si un joueur percute une de ces caisses, le joueur est bloqué pour un petit instant, comme pour les missiles.

Finalement, l'accélérateur modifie directement l'accélération maximale de la voiture qui l'utilise, ainsi la limite est repoussée et la constante qui est ajouter à l'accélération lorsque l'on appuie sur la touche avancer et elle aussi augmentée. Pour résumer, la vitesse maximum est augmenter et l'accélération aussi.

2.8 MonoJoueur - MultiJoueur [baudro_p]

2.8.1 Leurs Rôle

Le mode MonoJoueur tout comme le mode MultiJoueur permet à l'utilisateur de faire ce pour quoi ce "jeu" existe, c'est-à-dire jouer. Il permettent donc de jouer, sur

la carte de son choix qui peut être pécédemment créée avec l'éditeur de carte, avec ou sans bot, tout seul ou avec un deuxième joueur humain, sachant que le nombre de joueur maximum est de quatre.

Les rêgles du jeu, comme prendre tous les checkpoints, finir les tours,... y sont bien évidemment appliquées, et les quelques rêgles liées au moteur physique aussi.

2.8.2 Principe

Le principe est assez simple une fois que tous les moteurs, ainsi que la structure générale du projet sont fait. Il va suffire de regrouper toutes les informations données par le joueur, puis les traiter et ensuite les afficher, tout ce travail va être effectué par une boucle, c'est la boucle principale du jeu mais elle est aussi la plus simple.

Dans sa première version, elle était différente suivant le nombre de joueur humain et ordinateur, mais une généralisation a été effectuer pour lui permettre de s'adapter aux différentes configurations possibles grâce a des paramètre que l'on lui fait passer.

Elle se décompose comme ceci :
-elle va s'exécuter tant que le joueur n'aura pas dit explicitement de la quitter en appuyant sur la touche échappe
-le moteur physique va calculer les nouvelles positions de chaque objet en leurs appliquant les modifications liées aux collisions, aux déplacements des joueurs (touches appuyées),...
-vérification des rêgles du jeu (victoire, récupération des checkpoints,...)
-le moteur graphique va se charger de tout afficher à leurs bonne positions.

Avant cette boucle du jeu il y a une petite procédure d'initialisation, qui concerne la création de chaque joueur, c'est-à-dire leur positionnement sur la case du départ, la réinitialisation de leurs objets,... puis le chargement de la carte, et des objets présent sur celle-ci, et d'autre petits détails propre à chaque partie.

De même après la boucle du jeu ce trouve la désactivation de certain fonctions inutile autre part que dans une partie et le "nettoyage" des diverses choses inutiles.

cf. MonoMulti1

2.9 Effets Visuels

2.9.1 Lumières [auriau_f] [cadill_j] [baudro_p]

La lumière est un effet plus que primordiale, c'est elle qui donne tout le relief aux modèles,... mais elles ne sont que peu souvent modifiables dans les jeux, alors pourquoi ne pas laissez l'utilisateur au moins choisir entre un jeu de nuit avec quelques spots ou en plein jour. Cette option est modifiable à partir du GUI juste avant de lancer la partie.

2.9.2 Brouillard [auriau_f] [cadill_j] [baudro_p]

Le brouillard peut donner un effet plutot sympathique si il est utilisé avec parcimonie, de plus cet outil est très facile à utiliser sous OpenGL donc pourquoi s'en priver. Le brouillard permet d'estomper les détails qui sont lointain, donner une ambience plus calme, qui fasse moins agreessive de par les couleurs trop vives.

2.9.3 Caméra [baudro_p]

La caméra qui suis les voitures a été doté d'un "temps de retard" qui permet de voir la voiture qui commence à tourner avant que la camera ne tourne elle même. Il y a donc deux angles différents, un pour la voiture et un pour la caméra, celui de la caméra n'est pas remis à jour quand le joueur tourne sauf si la différence entre les deux angles est supérieur à un maximum fixé. Il continue de ce remettre à jour après, tant que les deux angles ne sont pas en phase.

Cette méthode permet d'avoir une caméra qui se remet automatiquement derrière la voiture même si l'on change brutalement l'angle d'accélération de celle-ci, comme par exemple lors de collision avec d'autre joueur, quand un joueur passe sur une caisse explosive ou est touché par un missile il se met à tourner pendant une seconde ce qui modifie son angle d'accélération.

2.9.4 Le compte à reboure [baudro_p]

Pour éviter que la course commence directement après avoir cliqué sur jouer, un compte à rebour laisse le temps au joueur de se préparer au départ. La durée de celui-ci est de trois secondes, et bien évidemment les joueurs ne peuvent pas bouger...

Ce compte à reboure est basé sur une boucle qui ne fait que afficher les décors et les objets ainsi que le temps qui défile, les commande n'y sont pas gérées, donc les joueurs sont bloqués tant que la boucle n'est pas finie. La condition de sortie de la boucle est que "l'heure" courante soit suppérieur à "l'heure" à laquelle la boucle a commencé plus

trois secondes. Le temps qui défile est la partie entière de l'heure de départ plus trois secondes moins l'heure actuelle. Une fois que la boucle est terminée, on entre dans la boucle principale du jeu.

2.10 Modèles 3D [baudro_p]

Les modèles 3D permettent de doner du volume à notre jeu, se sont eux qui sont affichés et qui représentent tout ce que le joueur fais. Ils permettent de le guider grâce au modèle du circuit, de ce voir avec son personnage qu'il a choisi et de visualiser les pièges laissés sur le sol par ses adversaires ou par lui même.

Pour le circuit chaque type de case a deux modèles différents pour la représenter, ce choix est accessible dans le l'éditeur de carte. Il y a deux modèles pour les voitures et quelques modèles pour les objets que l'on peut lancer et lacher par terre.

La skybox a été complètement refaite au cours du projet pour coller avec le style du jeu et donner un peu de relief au paysage.

La majorité des textures sont faite "maison" mais d'autre comme pour la skybox ou le sol on été récuperées sur le web ou dans des jeux. Par contre tous les modèles 3D sont fait "maison" et ça se voit...

cf. Modele3D1 - Modele3D2

2.11 Mode réseau [baudro_p] [auriau_f] [cadill_j]

2.11.1 Rôle du mode réseau

Le mode réseau de notre jeu permet à plusieurs utilisateurs (les clients) se jouer les uns contre les autres via un serveur, qui est lui-même joueur. Le réseau est intégré au GUI.

2.11.2 Avancement du mode réseau

Nous avons débuter la création du mode réseau pour la 3ème soutenance. Pour cela nous avons commencé la création d'un petit chat (salon de discussion), avec les composants delphi TclientSocket et TClientServer. Nous allons détailler leurs fonctionnements.

- Client :

Le client est le programme dont l'utilisateur va se servir pour envoyer des messages dans un salon de discussion. Avant de commencer à tchater l'utilisateur doit tout d'abord entrer l'adresse IP, ainsi que le port du serveur auquel il veut se connecter. Une fois la connection acceptée par le serveur, l'utilisateur peut envoyer du texte qui sera vu par tous les utilisateurs connectés au serveur. Le composant TClientSocket nous a simplifié la création du chat car il comporte de nombreuses fonctions intégrées simples à utiliser. Voici quelques procédures, brièvement expliquées, qui font partie du programme client.

Cette première procédure est associée à un bouton sur notre Form delphi, elle permet d'envoyer le texte contenu dans un Edit(dans notre cas Edit3) au serveur. La procédure **ClientSocket1.Socket.SendText(Edit3.Text)** prend donc en paramètre un string qui sera envoyé au serveur. La procédure**Memo1.Lines.Add('Informations envoyées au serveur')** sert uniquement à informer l'utilisateur que son message a été envoyé correctement.

Cette deuxième procédure **procedure TForm1.ClientSocket1Read(Sender : TObject ; Socket : TCustomWinSocket)** est en fait un événement du composant TClientSocket, elle permet de récupérer les messages envoyés par le serveur. La procédure **Socket.ReceiveText** renvoit le string qu'elle a reçu du serveur et enfin la procédure **Memo1.Lines.add (Socket.ReceiveText)** permet d'afficher le string reçu dans un Memo (dans notre cas Memo1).

- Serveur :

Le serveur sert de relais aux utilisateurs qui veulent communiquer. Le serveur va recevoir tous les messages envoyés par les clients et les transmettre à tous les autres clients connectés. Le serveur doit être lancé sur un port au choix. Il faut tout de même faire attention à ne pas le lancer un port déjà utilisé par d'autres programmes (ex : Navigateur web sur le port 80 , ou encore serveur ftp sur le port 21 ...). Le serveur marche aussi bien en réseau local que sur internet, si on veut accepter des connections entrantes depuis internet il faut ouvrir son firewall sur le port du serveur. Voici la procédure principale, brièvement expliquée, qui compose le serveur.

La **procedure TForm1.ServerSocket1ClientRead(Sender : TObject ; Socket : TCustomWinSocket)** est un événement du composant TServeurSocket, elle permet de récupérer les messages envoyés par les clients grâce à la fonction **Socket.ReceiveText** qui renvoit un string. Une fois le texte, envoyé par le client, récupéré, le serveur va l'envoyer à tous les clients connectés grâce à une boucle For qui part de zéro pour aller jusqu'au nombre de clients connectés moins un. La fonction permettant de récupérer le nombre de clients connectés est **ServerSocket1.Socket.ActiveConnections**

(dans notre cas "ServerSocket1"). Une fois rentré dans la boucle la procédure **ServerSocket1.Socket.Connections[I]. SendText(Socket.RemoteHost + ' dit : '+ b)** permet d'envoyer à chaque clients (chaque clients à un numéro I) le texte reçu précedemment. La procédure **Socket.RemoteHost** permet de récupérer le nom de l'ordinateur qui vient d'envoyer un message.

Bien évidemment le serveur et le client sont dotés d'une gestion d'erreur. Notamment si les adresses ip entrées sont non valide ou si le serveur tente d'envoyer des données à un client non connecté. La gestion d'erreur sera par la suite améliorée.

Création d'un buffer :
Pour la création du chat nous avons utilisé des fonctions qui nous permettées d'envoyer uniquement du texte. Pour notre jeu nous allons avoir besoin d'envoyer plusieurs types de données à la fois. C'est pourquoi nous allons utiliser un buffer. Voici un exemple de buffer de type trame.

Notre buffer est un enregistrement qui peut contenir plusieurs données.

- Client :

Pour envoyer un buffer le principe est à peu prés le même que pour envoyer du texte. On va définir une variable (dans notre cas "buffer") de type trame, puis on va remplir chaque champ de cet enregistrement avec les valeurs que l'on souhaite envoyées. Ensuite nous allons stocké la taille de ce buffer dans une variable de type integer (dans notre cas "a") en appelant la fonction **sizeof(buffer)**. Et enfin pour envoyer notre buffer nous allons utiliser la procédure **ClientSocket1.Socket.SendBuf(buffer,a)**(dans notre cas Socket1) qui prend en paramètres un buffer et sa taille.

Pour recevoir un buffer le principe est lui aussi équivalent à celui du chat (on utilisera la même procédure que pour le chat " **procedure TForm1.ServerSocket1ClientRead(Sender : TObject ; Socket : TCustomWinSocket)**"). On récupère tout d'abord la taille du buffer reçu grâce à la fonction **socket.ReceiveLength**. Puis on va tester si cette taille est inférieure ou égale à la taille maximum d'un buffer. Puis on utilise la fonction **socket.ReceiveBuf(b,taille) ;** ("b" est une variable de type trame et "taille" correspond à la taille du buffer reçu) pour récupérer le buffer réçu. Le buffer est ainsi stocké dans la variable "b". Cf client chat

- Serveur :

Tout comme pour le serveur du chat on va utiliser la **procedure TForm1.ServerSocket1ClientRead(Sender : TObject ; Socket : TCustomWinSocket)**. Comme pour le client on récupère la taille du buffer reçu toujours avec la même fonction et on teste si la taille du buffer est inférieure ou égale à la taille maximum d'un buffer et on stocke le

buffer reçu dans une variable de type trame (voir client ci-dessus). Ensuite on va se servir de la même boucle que pour le serveur du chat. A la difference que la fonction d'envoie va être **ServerSocket1.Socket.Connections[I].SendBuf(Buffer,taille)** ("Buffer" est le buffer reçu par le serveur et "taille" est la taille du buffer reçu).Cf serveur chat.

Envoi de coordonnées :

Grâce à la réalisation du buffer, nous avons réalisé un serveur et un client comportant chacun une fenêtre OpenGL. Chaque fenêtre OpenGl comporte un carré, il est possible de faire bouger le carré grâce au programme client. Le client va envoyer au serveur tous les déplacements que le carré va faire, ainsi la fenêtre OpenGL du serveur affichera aussi les déplacements du carré. Cf cube.

Pour l'intégration du mode réseau dans notre jeu, nous avons conservé la méthode qui utilise un buffer. Notre buffer contient ainsi la position de la voiture , son angle , son accélération .... De plus le buffer peut contenir beaucoup d'autre informations , le classement du joueur, le nombre de tour restant ainsi que le type d'objet en possession de la voiture.

Le joueur peut ainsi choisir d'être client ou serveur, le serveur choisi la map sur laquelle les autres joueurs vont jouer. Une fois que tous les joueurs sont connectés le serveur peut décider de lancer la partie sur tous les pc en même temps. Aprés que la partie se soit lancer sur tous les pc, les régles du jeu restent les mêmes que dans les autres modes de jeu. Cf Serveur et client.

2.12 Gestion du son [besson_p]

2.12.1 Introduction

Cette Partie du projet pourtant plus lègère en terme de reflexion algoritmique doit cependant etre realisée avec la même attention. Après l'affichage 3D, le son est la partie la plus visible du projet et ne doit pas être négligée. Dans notre projet, un jeu de voiture, il faut faire correspondre des évènements comme des collisions ou le bruit provenant du moteur avec des sons adéquats. De plus il faut une musique d'ambiance pour le GUI, puis pendant la partie et lors de la victoire d'un des joueurs. La partie concernant la musique à été traitée de manière telle que les utilisateurs du programme puissent écouter leur propres fichiers sonnores (mp3, wav, etc).

2.12.2 Choix de la Librairie

Pour le son nous utilisons la librairie **FMOD** téléchargeable gratuitement sur le site *http ://fmod.org/*. Nous avons préféré utiliser cette dernière plutôt qu'une autre pour les raisons suivantes :

- Tout d'abord, celle-ci est simple d'utilisation et parfaitement fonctionnelle. Fmod n'utilise pas des milliers de fonctions et est malgré tout très performante. Elle permet en effet de lire très simplement des sons, que ce soit en .mp3, .wav ou autres et de leur donner divers effets qui peuvent être très intéressants pour un jeu en 3D comme le nôtre. Et ce avec un rendu assez réaliste. - Ensuite, la dll est très légère, seulement 150 Ko par rapport aux autres dll utilisées par le jeu et même par rapport aux autres dll de gestion des sons.
- Ensuite, la dll est très légère, seulement 150 Ko par rapport aux autres dll utilisées par le jeu et même par rapport aux autres dll de gestion des sons.
- Enfin fmod fonctionne sur n'importe quelle plate-forme. Même si en prepa notre projet n'est pas sensé fonctionner sur d'autres plate-formes que Windows, cela nous a paru être preuve de bonne qualité.

2.12.3 Fonctions Principales

Pour commencer, l'initialisation de FMOD avec les paramétrages de lecture des fichiers. Grâce à la procedure pInitFMOD que nous avons codé, nous choisissons le type de spatialisation du son, la fréquence d'échantillonnage et le nombre de canaux désirés pour l'application. Ces informations peuvent dans un souci de compatibilité être changées dans le menu principal du jeu, tout comme la résolution d'affichage par exemple.

Nous avons codé trois fonctions de base qui permettent de lire, de pauser et de stopper la lecture d'un fichier (en fait en utilisant le type TSound défini plus bas). Dans un même temps, le statut change d'état selon le type énuméré suivant : Voir annexe pour le code.

Ainsi nous savons si un fichier est en cours de lecture ou non. Nous pouvons par exemple attendre la fin de la lecture d'un fichier pour en débuter la lecture d'un autre.

La fonction pListDir codée par nos soins parcours un répertoire donné en paramètre à la recherche de fichier d'un type défini, ici des fichiers musicaux. Puis nous remplissons un vecteur contenant des éléments du type Tsound que nous avons codé de la manière suivante :

Nous pouvons définir dans l'initialisation de la librairie un nombre de cannaux de lecture nous permettant de gérer parfaitement notre liste de lecture en complément des états (TMusicState). Chaque morceau ne peut pas être lu plusieurs fois simultanément, et le nombre de fichiers lus simultanément est connu.

2.12.4 Les Bruitages

Au lancement du programme, un vecteur d'enregistrement est rempli grâce à la procédure pListDir avec les sons destinés aux bruitages. Ces sons sont statiques, ils doivent être tous présents et avoir un nom leurs permettant de garder leur ordre (classment) dans le répertoire. En somme il ne faut pas y toucher ils font partis des fichiers du jeu.

Nous avons donc à ce stade de l'execution du programme, un vecteur d'enregistement renseigné et nous sommes capable de lire chacun de ces fichiers.

Tous les événements du jeu tels que l'accélèration ou les collisions proviennent du type *voiture* qui est un enregistrement de toutes les données concernant le déplacement des bolides, implémentés dans le moteur physique. Voici un extrait du type *voiture* :

Chaque champ d'enregistrement contient l'information nécessaire à la lecture d'un son. Le type de véhicule, l'accélèration, la possession d'un item ainsi que la vitesse. Selon des valeurs d'acceleration un son différent sera joué. Initialement, et lorsque le jouer ne possède pas d'item le champ correspondant (de type string) vaut la chaine de caractère vide. Nous sommes donc indépendants des touches du clavier et tout passe par le type *voiture*.

2.12.5 La Musique

Pour ce qui est de la musique, nous procèdons de même manière que pour les bruitages si l'utilisateur ne souhaite pas ecouter ses propres morceaux musicaus. En effet, un repertoire destiné exclusivment aux fichiers sons de l'utilisateur existe. S'il est vide ou contient un fichier non valide, le programme va utiliser les fichiers par défaut. Pendant la partie, une musique d'ambiance pouvant être arrêtée ou reinitilisée et pendant le GUI un autre.

Par contre, si l'utilisateur souhaite ecouter ses fichiers, le répertoire de destination de ses fichiers étant rempli (au moins un fichier)la procédure pListDir va scanner le repertoire et créer une liste de lecture. Pendant l'execution du jeu, l'interface permet l'affichage de la musique jouée. Cette playlist se manipule avec des fonctions telles que : mute, pause, prec, suiv et stop.

2.12.6 Conclusion

Débuté et en partie achevé, en ce qui concerne les algorithmes, pour la 3ème soutenance, le son est dorénavant parfaitement operationel et rempli toutes les fonctions ennoncées dans le cahier des charges. L'accès à la personnalisation des listes de lecture est un atout pour notre projet. Même si l'idée du repertoire alloué à 'sa' playlist est reprise d'un cellèbre jeu de voiture son application est propre à notre jeu.

2.13 Site web [cadill_j] [auriau_f]

Parallèlement au développement du jeu, nous avons crée un site web qui a eu pour but de présenter le projet aux internautes. Son adresse est : http ://lazit.free.fr

2.13.1 Rôle du site web

Sur ce site, on trouve le cahier des charges, les différents rapports de soutenance, ainsi que les versions bêta et la version finale du jeu. Dans la rubrique liens, on trouve également une liste de bonnes adresses concernant le développement delphi/OpenGL. Une gestion des news a été implémentée pour suivre en direct l'avancement du projet. Pour finir, il est possible de contacter chaque membre du groupe via le site.

2.13.2 Design

Le design du site ressemble fortement à celui du jeu, c'est à dire des couleurs vives, rappelant l'univers " cartoonesque " dans lequel nous nous sommes plongés.

2.13.3 L'avancement du site web

Lors de la première soutenance, le site était quasiment dans sa version finale. Il restait seulement à intégrer la gestion des news. Cf Capture news.

Nous devions présenter le site web pour la deuxième soutenance, ce qui a été réalisé. La gestion des news a été implémentée, grâce à une base de donnée MySql, gérée par EasyPhp. Cf Site.

2.14 Assemblage général du projet [auriau_f] [baudro_p] [besson_p] [cadill_j]

2.14.1 Installeur et désinstalleur

A l'aide d'un programme, nous avons créé un installeur et un désinstalleur. Le fichier setup.exe se lance directement à l'insertion du CD, à l'aide d'un fichier autorun.inf.

2.14.2 Réunification des unités [baudro_p] [besson_p]

2.14.3 Manuel d'installation et d'aide [auriau_f] [cadill_j]

Nous avons réalisé un manuel d'installation au format papier qui est fourni dans le boitier du jeu ainsi qu'une deuxième aide qui est inclus avec l'installation. L'aide qui est incluse sur notre CD-ROM est au format "HLP". Les fichiers au format HLP peuvent être lu sur tous les pc ayant pour système d'exploitation créé par Microsoft. Les deux aides réalisées, expliquent clairement comment installer le jeu, ainsi que l'utilisation complétes de toutes les options du jeu.

Conclusion

Pour conclure, nous sommes relativement content de nous, dans la mesure où nous avons atteint l'objectif que nous nous étions fixé au départ, c'est à dire mener ce projet à terme. Aussi, nous sommes heureux d'avoir terminé ce projet à 4, ce qui malheuresement n'est pas toujours le cas. Ce fut surtout une très bonne expérience, aussi bien au niveau de l'acquisition de nombreuses connaissances qu'au niveau de la réalisation d'un travail en groupe. En effet, il n'est pas toujours simple de faire abstraction des différences qui opposent les membres d'un groupe, dans le but de travailler le plus éfficacement possible.

Annexes

```
glBegin(GL_QUADS); glVertex3f(1.0,1.0,0.0);
glVertex3f(0.0,1.0,0.0); glVertex3f(0.0,0.0,0.0);
glVertex3f(1.0,0.0,0.0); glEnd();

Qui dit cube dit 6 faces rectangulaires.

glClear(GL_COLOR_BUFFER_BIT); glClear(GL_COLOR_BUFFER_BIT
|GL_DEPTH_BUFFER_BIT); glMatrixMode(GL_MODELVIEW);
gluLookAt(5,5,5,0,0,0,0,1,0); glBegin(GL_QUADS);
glVertex3i(1,1,1); glVertex3i(1,-1,1); glVertex3i(-1,-1,1);
glVertex3i(-1,1,1); //1 face

\ldots

glVertex3i(-1,-1,-1); glVertex3i(-1,-1,1); glVertex3i(1,-1,1);
glVertex3i(1,-1,-1); //6 faces glEnd(); SwapBuffers(DC);
```

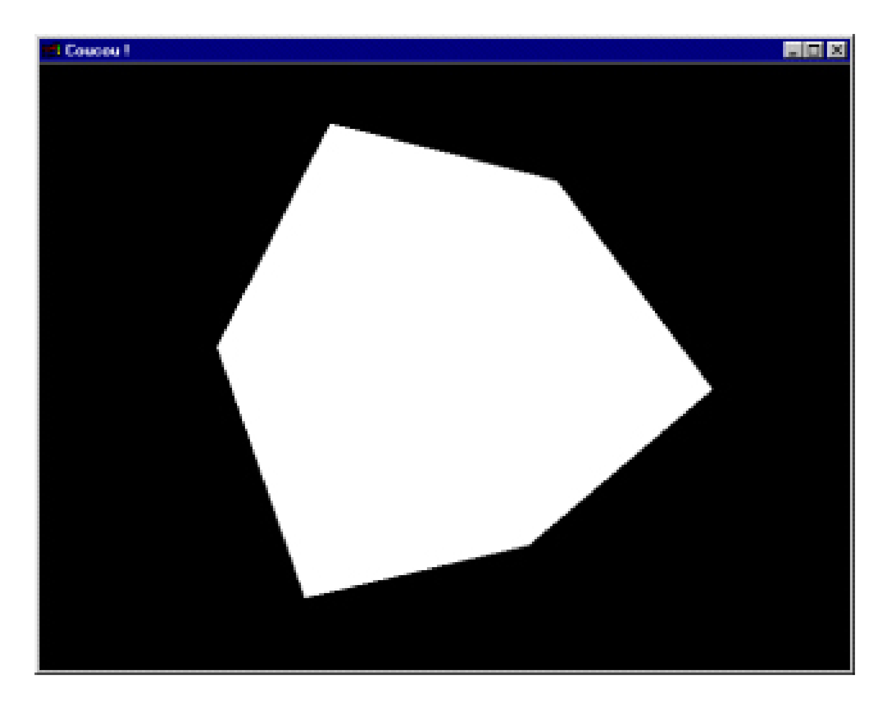


Fig. 1 – Cube blanc

```
glClear(GL_COLOR_BUFFER_BIT); glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT); glMatrixMode(GL_MODELVIEW);
gluLookAt(5,5,5,0,0,0,0,1,0); glBegin(GL_QUADS); glColor3d(1,0,0);
glVertex3i(1,1,1); glVertex3i(1,-1,1); glVertex3i(-1,-1,1);
glVertex3i(-1,1,1);//1 face

\ldots

glColor3d(1,0,1); glVertex3i(-1,-1,-1); glVertex3i(-1,-1,1);
glVertex3i(1,-1,1); glVertex3i(1,-1,-1);//6 faces glEnd();
//"Cette fois, j'le sens bien\ldots" SwapBuffers(DC)
//glutSwapBuffers(); pour glut
//glutPostRediplay(); Uniquement pour GLUT
```

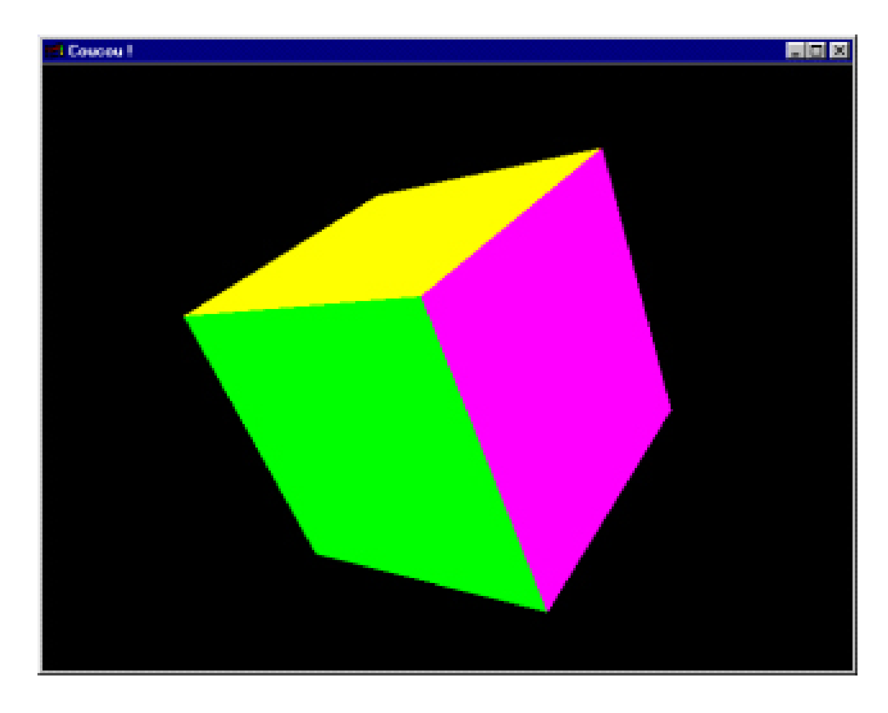


FIG. 2 – Cube couleur

```
for i := 0 to 8 do
  for j := 0 to 8 do
          begin
            glPushMatrix;
            glTranslated(i*10+5, j*10+5, 0);
            glRotate(90, 1, 0, 0);
            AffModel(13);
            glPopMatrix;
          end;
for i := (((realToInt(Tvoiture[1].posX)div 10))-2)   to
(((realToInt(Tvoiture[1].posX))div 10)+2) do
  for j := ((((realToInt(Tvoiture[1].posY))div 10))-2) to
  (((realToInt(Tvoiture[1].posY))div 10)+2) do begin
      if (i >= 0) and (i < 8) then
        if (j >= 0) and (j < 8) then begin
            glPushMatrix;
            glTranslated(i*10+5, j*10+5, 0);
            glRotate(90, 1, 0, 0);
            AffModel(mapModel[i+1][j+1]);
            glPopMatrix;
          end;
    end;
end;
```

```
Model[1]:=T3DModel.Create; Model[1].loadfromfile('.\hummer.3ds') ;
Model[1].Draw ;
```

```
p_load_model(model:integer;x,y,z:real;model_fixe:boolean;rot:real);
```

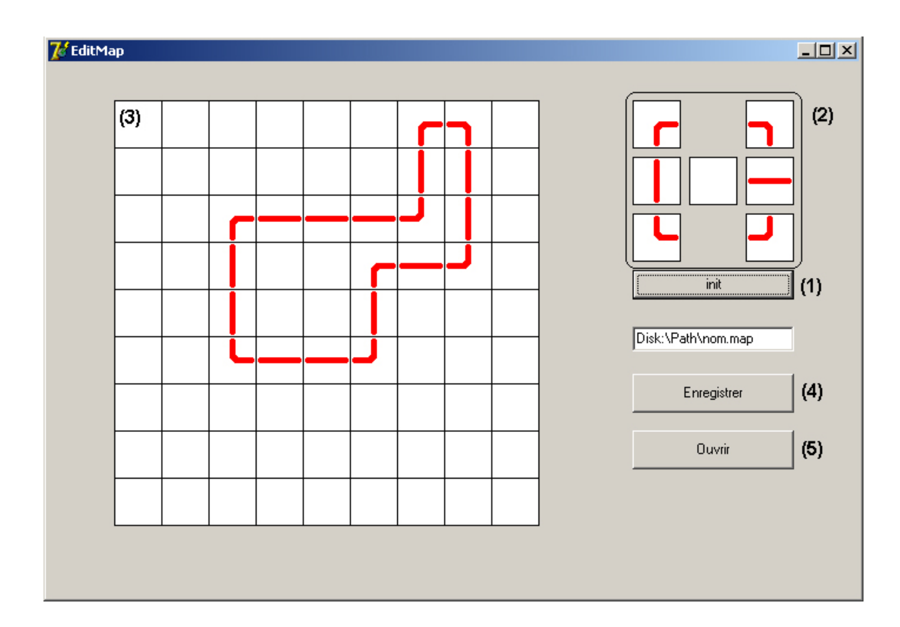


FIG. 3 – EditMap1

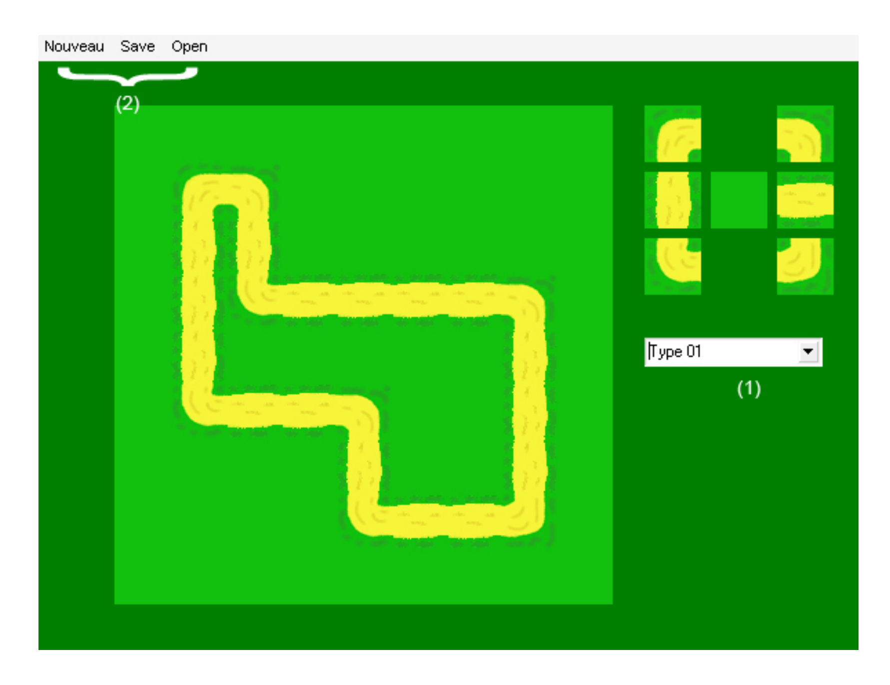


Fig. 4 – EditMap2

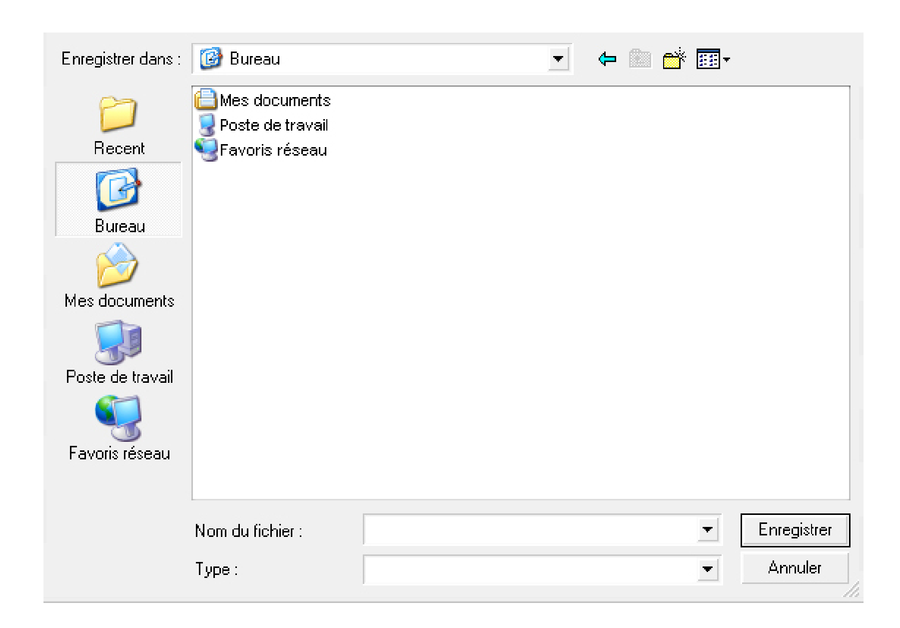


Fig. 5 – EditMap3

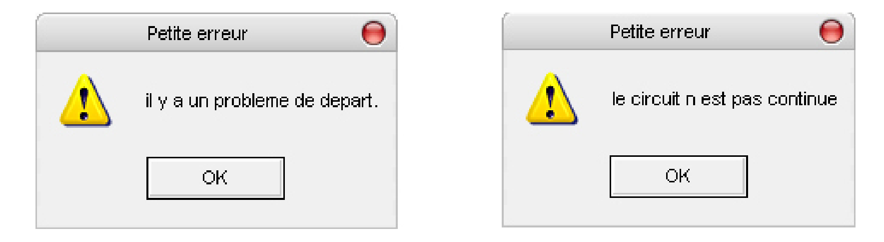


Fig. 6 – EditMap4

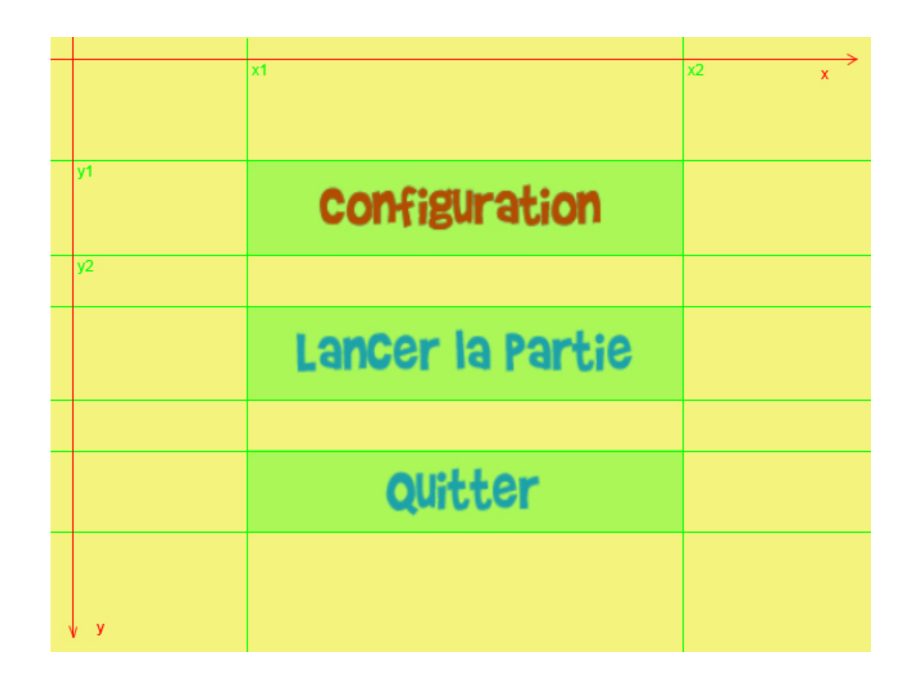


Fig. 7 – GUI1

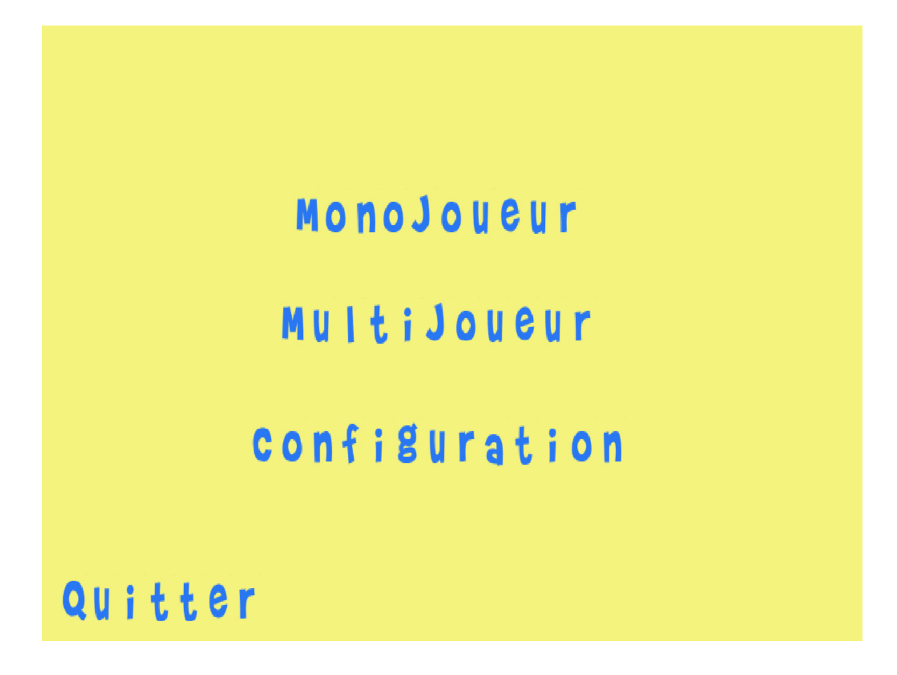


Fig. 8 – GUI2

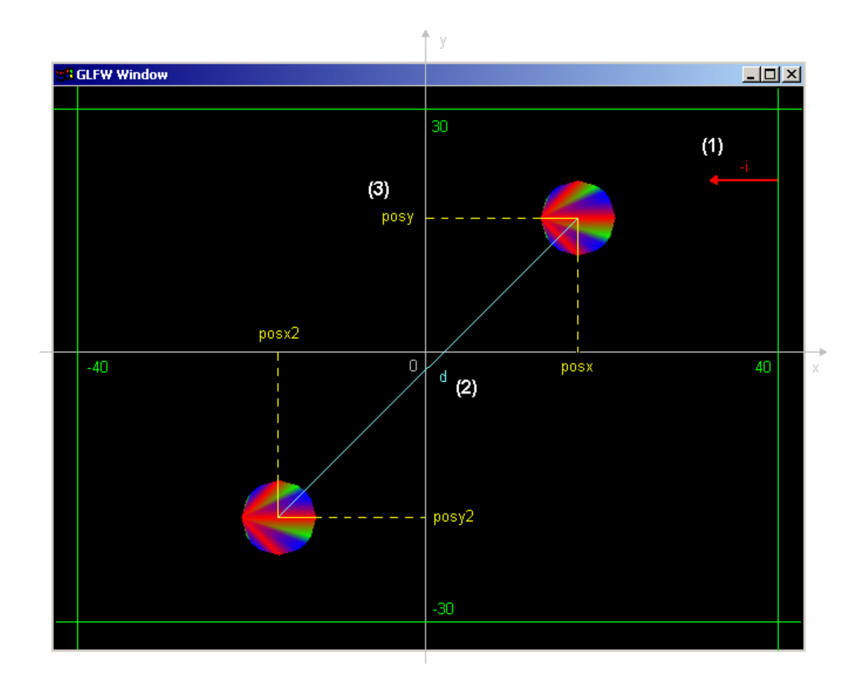


FIG. 9 – MoteurPhysique1

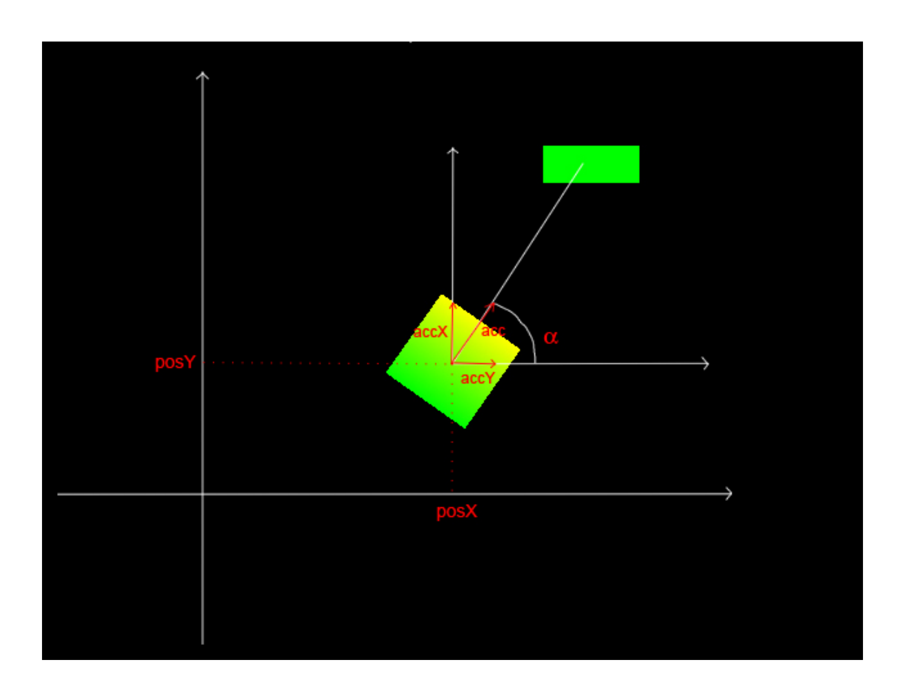


Fig. 10 – MoteurPhysique2

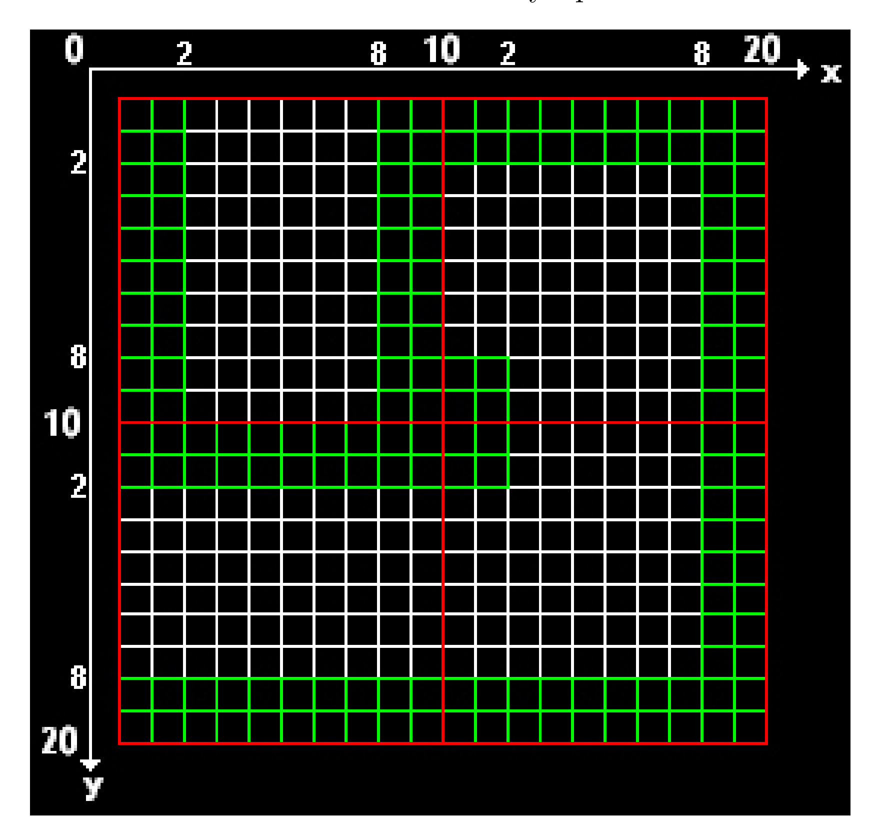


Fig. 11 – MoteurPhysique3

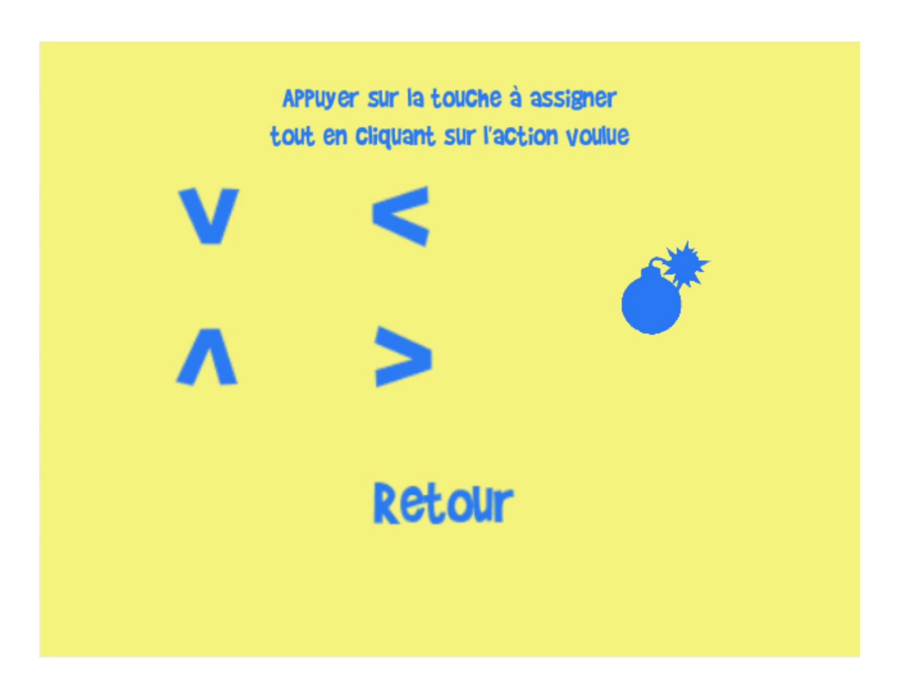


Fig. 12 – EntreesSorties1

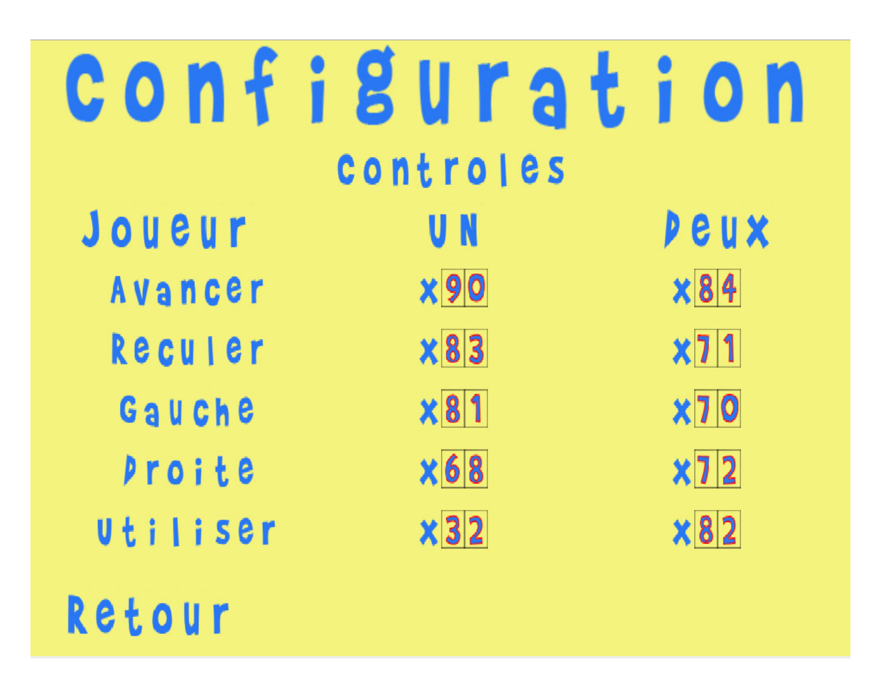


Fig. 13 – EntreesSorties2

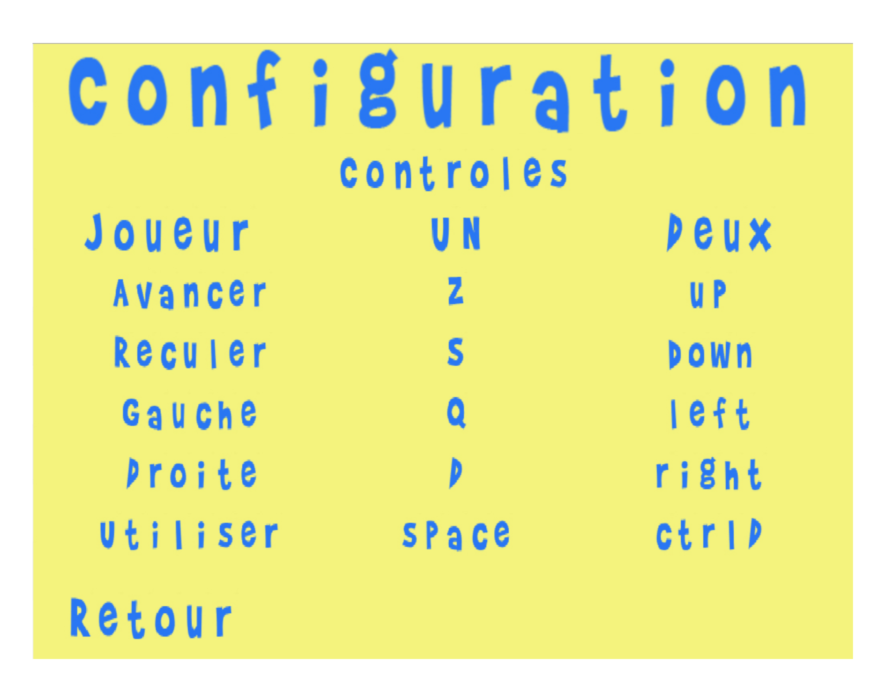


Fig. 14 – EntreesSorties3

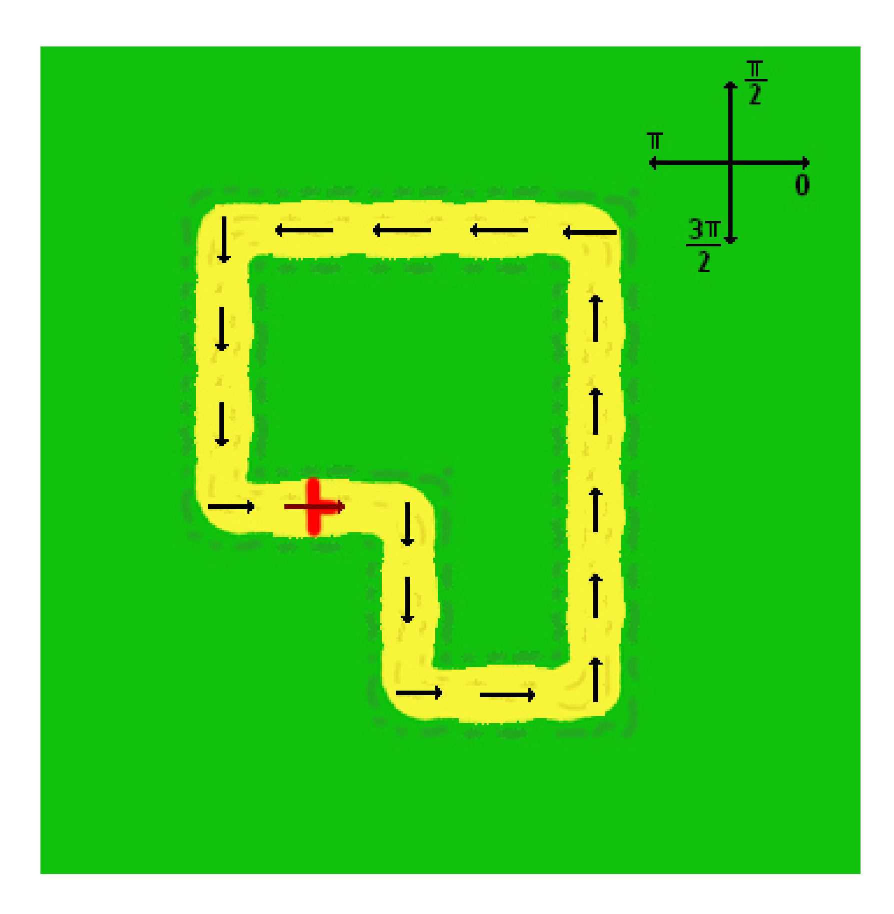


Fig. 15 – IA1

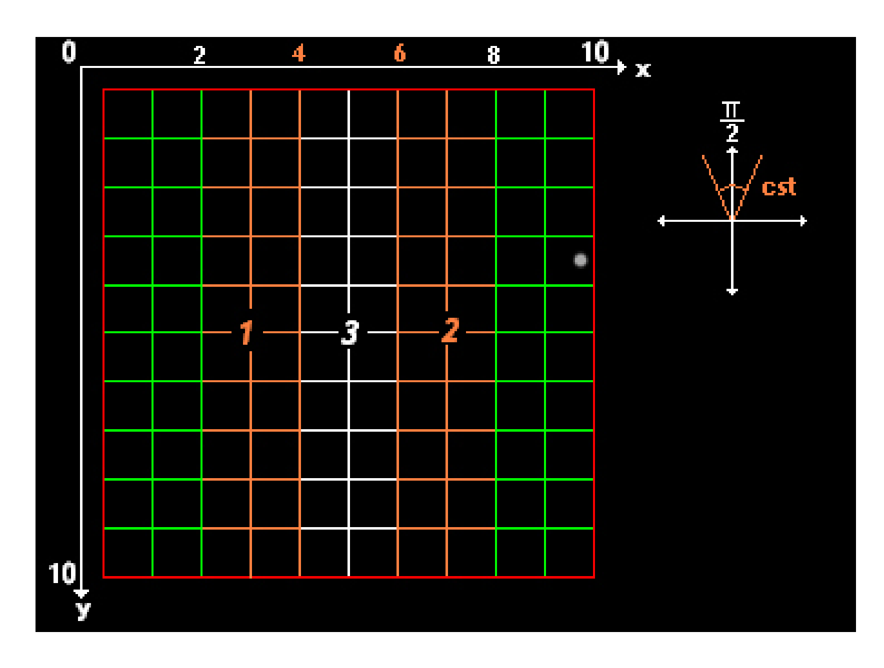


Fig. 16 – IA2

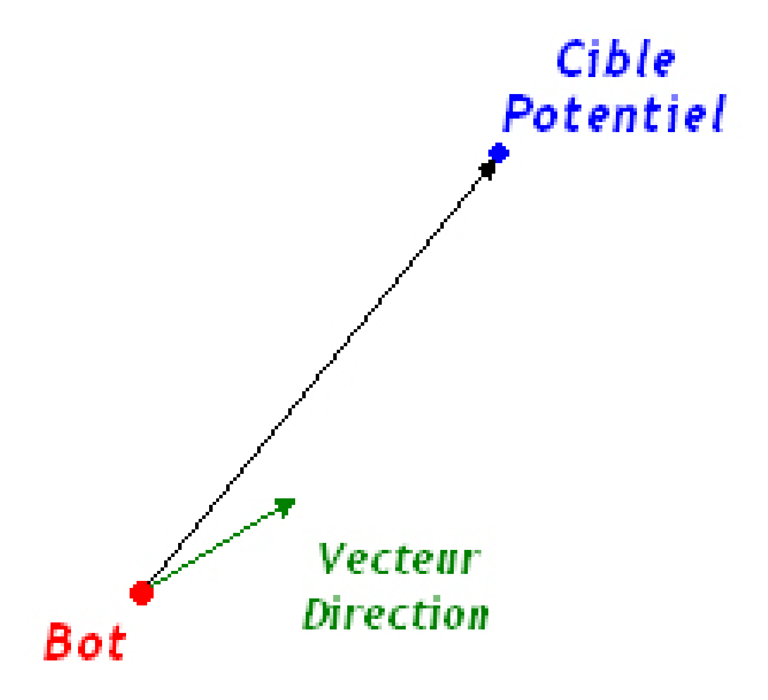


Fig. 17 – IA3



Fig. 18 – MonoMulti1

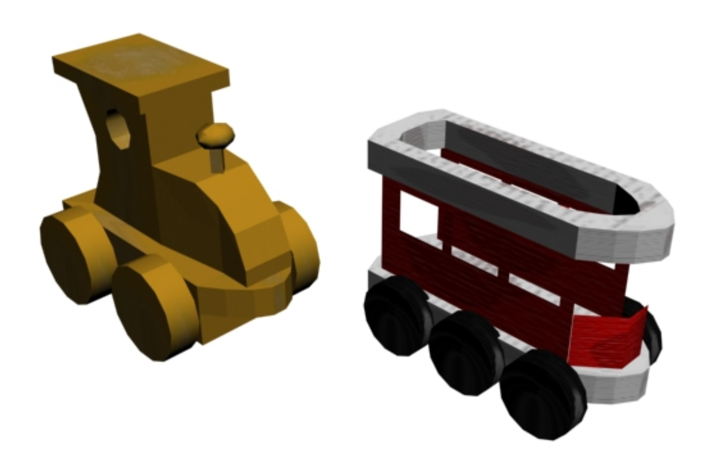


Fig. 19 – Modele3D1

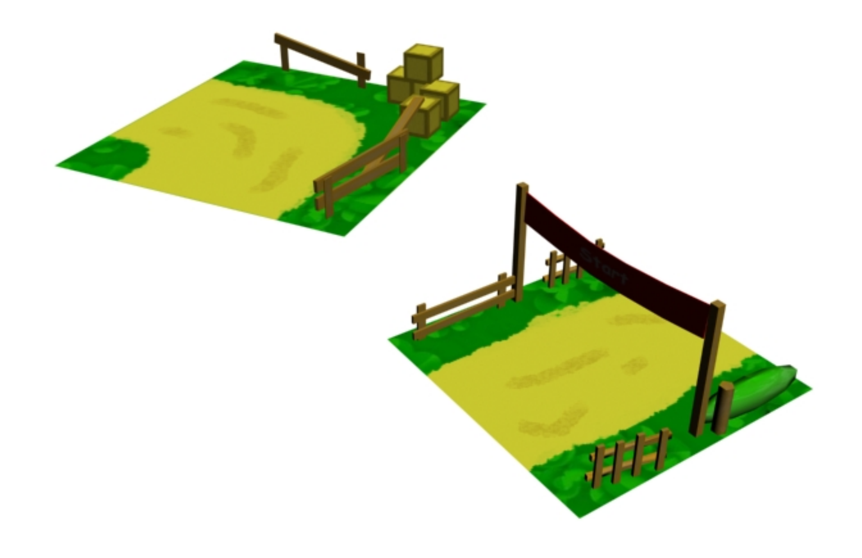


Fig. 20 – Modele3D2

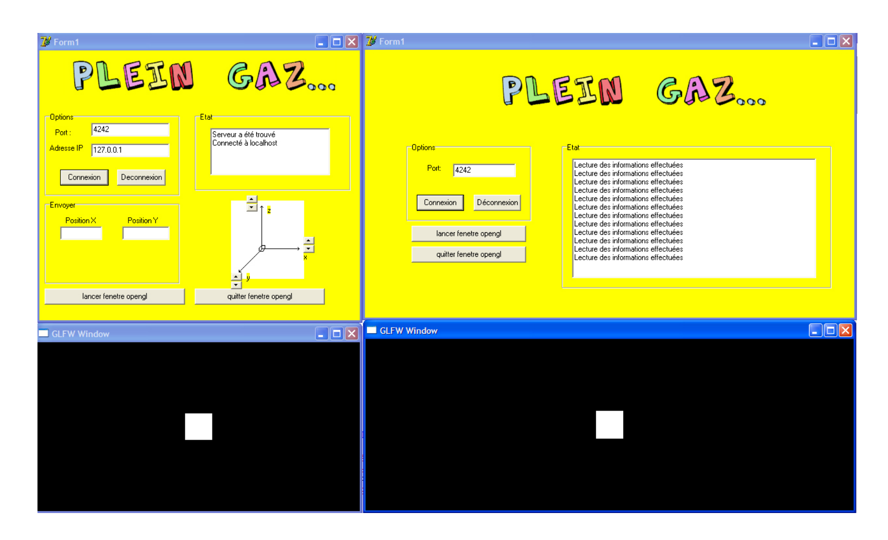


Fig. 21 – Cube

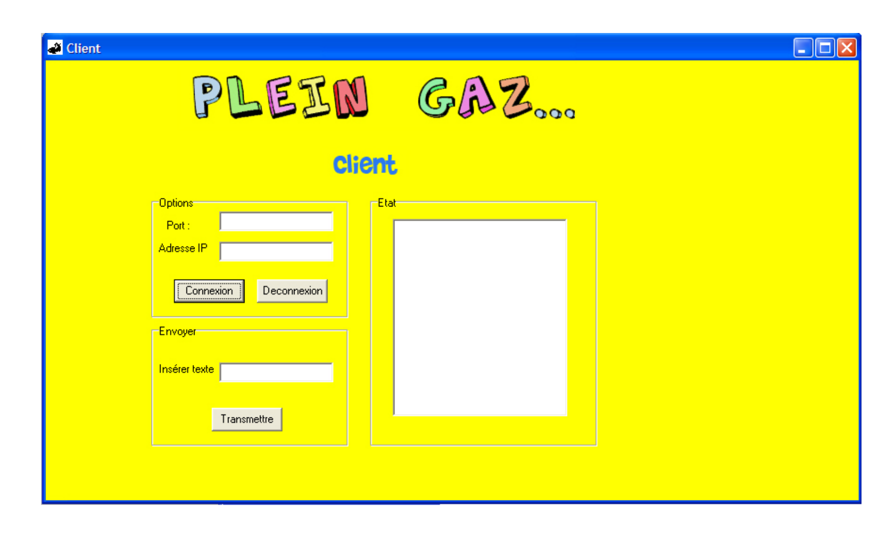


Fig. 22 – Client Chat



Fig. 23 – Serveur Chat

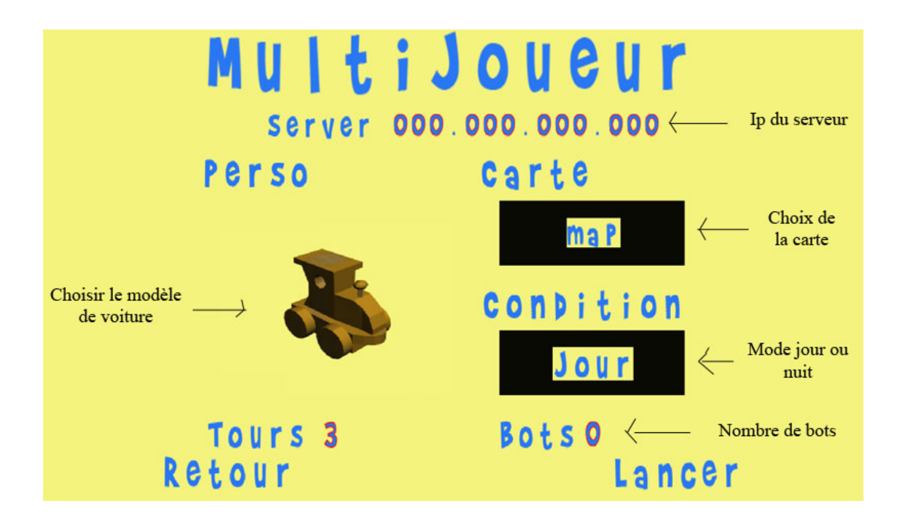


Fig. 24 – Serveur



Fig. 25 – Client

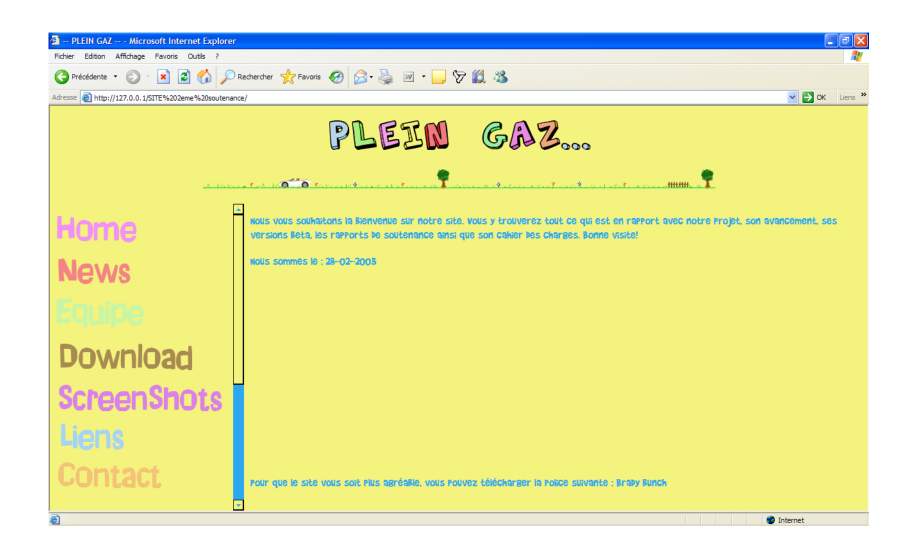


Fig. 26 – SITE



Fig. 27 – Capture News